The Waterfall of Liberty: Decoy Routing Circumvention that Resists Routing Attacks

Milad Nasr
University of Massachusetts Amherst
milad@cs.umass.edu

Hadi Zolfaghari
University of Massachusetts Amherst
hadi@cs.umass.edu

Amir Houmansadr
University of Massachusetts Amherst
amir@cs.umass.edu

ABSTRACT

Decoy routing is an emerging approach for censorship circumvention in which circumvention is implemented with help from a number of volunteer Internet autonomous systems, called decoy ASes. Recent studies on decoy routing consider all decoy routing systems to be susceptible to a *fundamental attack* —regardless of their specific designs—in which the censors re-route traffic around decoy ASes, thereby preventing censored users from using such systems. In this paper, we propose a new architecture for decoy routing that, by design, is significantly stronger to rerouting attacks compared to *all* previous designs. Unlike previous designs, our new architecture operates decoy routers only on the downstream traffic of the censored users; therefore we call it *downstream-only* decoy routing. As we demonstrate through Internet-scale BGP simulations, downstream-only decoy routing offers significantly stronger resistance to rerouting attacks, which is intuitively because a (censoring) ISP has much less control on the downstream BGP routes of its traffic.

Designing a downstream-only decoy routing system is a challenging engineering problem since decoy routers do not intercept the upstream traffic of censored users. We design the first downstream-only decoy routing system, called Waterfall, by devising unique covert communication mechanisms. We also use various techniques to make our Waterfall implementation resistant to traffic analysis attacks.

We believe that downstream-only decoy routing is a significant step towards making decoy routing systems practical. This is because a downstream-only decoy routing system can be deployed using a significantly smaller number of volunteer ASes, given a target resistance to rerouting attacks. For instance, we show that a Waterfall implementation with only a *single* decoy AS is as resistant to routing attacks (against China) as a traditional decoy system (e.g., Telex) with 53 decoy ASes.

CCS CONCEPTS

• **Security and privacy** → **Pseudonymity, anonymity and untraceability**; **Privacy-preserving protocols**;

KEYWORDS

Internet Censorship, censorship circumvention, decoy routing, routing attacks

1 INTRODUCTION

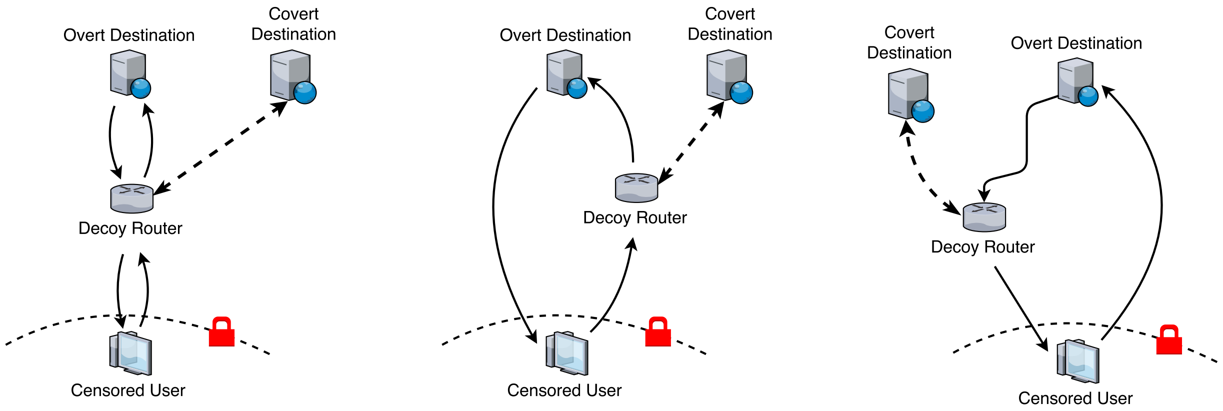
Internet censorship continues to remain a global threat to the freedom of speech, ideas, and information. An increasing number of repressive regimes and totalitarian governments implement Internet censorship [11, 32, 33, 46] using various techniques such as IP address filtering, DNS interference, and deep-packet inspection [31, 50]. To help the impacted Internet users bypass such censorship, various *circumvention systems* have been designed and deployed by academics and practitioners [5, 13, 17, 24, 30, 42].

Decoy routing [21, 25, 59] is an emerging approach for censorship circumvention. Unlike traditional circumvention systems (e.g., Tor [13], VPNs [42, 43], and Psiphon [24]) where circumvention software is implemented on computer servers, in decoy routing the circumvention software is mounted on the Internet routers of some volunteer Internet autonomous systems called *decoy ASes*. That is, decoy routing takes an end-to-middle proxying approach in contrast to the traditional end-to-end proxies. This, by design, makes decoy routing resistant to IP address blocking, which is one of the most effective, most widely practiced censorship mechanisms for disabling traditional end-to-end circumvention systems like Tor and VPNs.

Existing decoy routing systems are known to be susceptible to particular routing attacks by the censors, known as the routing around decoys (RAD) attacks [48]. In RAD, the censoring ISPs manipulate the upstream Internet (BGP) routes of their Internet users such that the censored users' traffic does not transit through any of the (identified) decoy ASes, therefore preventing the users from utilizing decoy routing systems for circumvention. Recent studies on decoy routing [22, 41, 48] consider RAD to be a *fundamental* weakness of the decoy routing approach —regardless of the technical specifications of decoy routing systems. In this paper, **we debunk this common belief** by proposing *a new decoy routing architecture that—by design—provides significantly stronger resistance to RAD and similar rerouting attacks compared to traditional decoy routing systems*. The *core idea* of our new decoy routing architecture is to operate decoy routers *only* on the downstream traffic of the censored users, with no need for intercepting the upstream traffic of the censored users; therefore, we name our new approach *downstream-only decoy routing*. This is in contrast to the architecture used by —all—previous decoy routing systems [4, 15, 21, 25, 58, 59] where they all need to intercept the upstream traffic of the censored users (some previous designs need to intercept both upstream and downstream [21, 25, 59], while others need to intercept only the upstream [4, 15, 58] traffic). Figure 1 illustrates the difference between various architectures.

CCS '17, October 30-November 3, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/10.1145/3133956.3134075



(a) Upstream (Type I) [21, 25, 59]

(b) Upstream (Type II) [4, 15, 58]

(c) Downstream-Only

Figure 1: Decoy routing architectures: In our downstream-only architecture shown in (c), decoy routers *do not need* to intercept upstream overt traffic, but just downstream overt traffic (note that the downstream traffic may or may not be intercepted by the decoy routers). By contrast, previous designs (shown in (a) and (b)) *need* to intercept upstream overt traffic to be able to operate (some previous work shown in (a) need to intercept both upstream and downstream).

As we demonstrate through Internet-scale BGP path simulations, downstream-only decoy routing offers significantly stronger resistance to routing attacks by the censors compared to the traditional (upstream) decoy routing approach. Intuitively, this is because a typical (censoring) ISP has significantly more leverage on shaping the BGP routes of its upstream traffic than its downstream traffic. Particularly, a censoring ISP can re-route its upstream traffic by applying per-destination routing rules (e.g., re-route only the traffic towards specific Internet destinations), however, it can *not* re-route downstream traffic by applying per-source rules (e.g., the censoring ISP has to re-route either all or none of its downstream traffic through each of its Internet provider ASes). The RAD attack, as studied in the literature [22, 41, 48], only modifies the upstream BGP routes, therefore does not trivially work against our downstream-only architecture. We therefore introduce and investigate two variants of the RAD attack that are specifically tailored to downstream-only decoy routing systems. We demonstrate through Internet-scale path simulations that applying such routing attacks on downstream-only decoy routing is *extremely costlier* than routing attacks on traditional (upstream) decoy routing systems. For instance, we show that a downstream-only decoy routing system implemented on only a *single* decoy AS is as resistant to routing attacks (by the Chinese censors) as a traditional decoy routing system (e.g., Telex) with 53 decoy ASes. We argue that **downstream-only decoy routing is a major step forward in making decoy routing systems practical**, as they need to be deployed on a significantly smaller number of volunteer ASes for a target resistance to routing attacks.

Note that designing a downstream-only decoy routing system is an *extremely challenging* engineering problem. All previous (upstream) decoy routing designs [4, 15, 21, 25, 58, 59] use the upstream traffic of the censored users to communicate essential covert messages to the decoy routers, such as registration requests, HTTP GET requests for blocked destinations, etc. This is not possible in a downstream-only decoy routing system since the upstream traffic of the censored users is not supposed to be intercepted by the decoy routers. We *design the first downstream-only decoy routing system, called Waterfall*.[1] We use a set of complementary mechanisms to enable low-latency upstream covert communications in Waterfall despite its downstream-only architecture. Particularly, we use HTTP redirection and several other techniques to enable real-time upstream covert communications in Waterfall. We also use various techniques to make Waterfall resistant to various traffic analysis attacks. We have built a **fully functional prototype** of Waterfall, which is publicly available online [55].

In summary, we make the following **main contributions**:

(1) We propose a novel decoy routing architecture, called downstream-only decoy routing, that by-design provides much stronger resistance to the infamous routing attacks by the censors, compared to previous (upstream) decoy routing systems. We demonstrate the superior resistance of downstream-only decoy routing against routing attacks trough extensive Internet-scale simulation of BGP routes.
(2) Designing a downstream-only decoy routing system is a challenging engineering problem. In this paper, we design *the first* downstream-only decoy routing system, called Waterfall, by using several novel covert communication techniques. While resisting routing attacks is the main focus of this paper, we also use novel techniques to make Waterfall resistant to traffic analysis attacks.
(3) We have built a fully functional prototype of Waterfall, which is publicly available online [55].

2 BACKGROUND

2.1 Major Censorship Circumvention Systems

Traditional censorship circumvention systems like Tor [12, 13, 28], VPNs [42, 43], Psiphon [24], and many other systems [5, 14, 30]

[1] It is hard to deflect a waterfall's falling water stream!

work by setting up circumvention *proxy* servers outside the censorship territories. Such proxies are used by censored users to bypass censorship by relaying the traffic of the censored users to their intended blocked Internet destinations. For instance, Tor [13] is comprised of several thousands of volunteer proxies, and every censored user will select a number of such proxies (usually three) to relay her traffic to censored destinations. Unfortunately, proxy-based circumvention systems are trivially blockable as soon as the censors identify the IP addresses of their proxies. For instance, Tor bridges [12] are discovered and blocked in-the-wild through various mechanisms [6, 16, 35, 56, 57] such as insider attacks, zig-zag attacks, and active probing.

Domain fronting [17] is a new approach to set up circumvention proxies that resist IP address filtering. In this mechanism, circumvention proxies, e.g., Tor meek bridges [38], are run as web services that share IP addresses with other, non-circumvention services, therefore blocking them will impose collateral damage to the censors. Domain fronting, however, is an expensive circumvention solution due to the bandwidth and CPU costs charged by the hosting web services [39]. CDNBrowsing [19, 61] is a similar circumvention solution in which censored users directly fetch censored content from the public content-delivery networks like Akamai that are hosting them. CDNBrowsing is much cheaper than domain fronting as it does not run any proxies, however, it can unblock only a limited class of censored websites [61].

2.2 Decoy Routing Circumvention Approach

Decoy routing is a recent approach for censorship circumvention [21, 25, 59] whose design is motivated by the ease of IP address blocking of traditional proxy-based circumvention systems, as discussed above. In decoy routing, censorship circumvention is implemented with help from a number of friendly Internet autonomous systems, called **decoy ASes**. Each decoy AS modifies some of its routers (e.g., its border routers) such that they deflect the Internet traffic of censored users to the blocked Internet destinations requested by the users; the routers implementing the deflection functionality are called **decoy routers**. By design, decoy routing defeats IP address blocking, which is the most effective mechanism used in-the-wild to disable traditional proxy-based circumvention systems like Tor and VPNs. Figure 1 illustrates the decoy routing approach.

To use a decoy routing system, a censored user will first need to establish an encrypted connection (e.g., a TLS connection) with a non-censored **overt** destination. The client selects the overt destination such that his network communications with that destination transit through at least one decoy router. A decoy routing system is composed of an **upstream** and a **downstream** covert communication channel through which the censored users communicate with the decoy routers intercepting their overt traffic. Particularly, the upstream covert channel is used by the censored users to send covert messages *to decoy routers*, e.g., HTTP GET requests for blocked Internet destinations. On the other hand, the downstream covert channel is used by the decoy routers to send covert messages *to the censored users*, e.g., the HTTP responses obtained from blocked Internet destinations. Each decoy routing system also has a **signaling** (or registration) channel through which a censored user informs the decoy routers intercepting her traffic of her willingness to use the decoy routing system (the upstream and downstream covert communications will establish only after a successful registration).

2.3 Existing Decoy Routing Designs

Table 1 compares the design of major decoy routing systems proposed in the literature, along with our system Waterfall. As can be seen, all previous designs use TLS records as their upstream and downstream covert communication channels in different fashions. Early decoy routing systems like Telex [59], Cirripede [21], and Curveball [25] simply embed upstream and downstream covert messages into the overt TLS connections man-in-the-middled by decoy routers. They encrypt the covert content with keys previously exchanged between decoy routers and censored clients during the signaling stage to foil deep-packet inspection by the censors. More recent designs perform a more sophisticated replacement of TLS records to provide stronger unobservability guarantees. For instance, Slitheen [4] embeds upstream covert content as a particular HTTP header in the upstream HTTP traffic towards an overt TLS website, and embeds downstream covert messages by replacing the leaf HTTP responses (e.g., images) from the overt website.

Previous designs use a variety of mechanism for the signaling channel. For instance, Cirripede uses steganography in TCP Initial Sequence Numbers (ISN) to register clients with the decoy routing system, and Telex and Slitheen use the TLS ClientHello random nonce field to signal decoy routers. Due to space constraints, we refer the reader to each of the design papers for further details.

2.4 Routing Around Decoys (RAD)

The routing around decoy (RAD) attack was presented by Schuchard et al. [48] as a generic attack on decoy routing systems. The main intuition behind the attack is that, for any given Internet destination, a censoring ISP is likely to know multiple upstream BGP routes. Therefore, the censoring ISP can discard the upstream BGP routes that transit through known decoy ASes, and instead re-route traffic through decoy-free routes, even if those routes are not the “best” BGP paths based on the BGP route selection criteria. The objective of the RAD attack is to prevent censored users from using decoy routing systems by preventing their traffic from transiting through decoy routers, i.e., by routing their traffic around decoys.

The RAD attack imposes various costs to the censoring ISPs. If a censoring ISP does not know any decoy-free routes to a particular destination, it will need to drop all traffic to that destination, making that destination *unreachable*. To reduce the fraction of unreachable destinations due to RAD, Schuchard et al. argue that the censoring ISPs controlled by the same government (e.g., all Chinese ASes) can share their decoy-free BGP routes among themselves; therefore, a censoring (say, Chinese) ISP with no decoy-free routes to a particular destination will tunnel its traffic to that destination through another censoring (Chinese) ISP who knows a decoy-free route to the target destination. Schuchard et al. [48] show that doing so will keep the fraction of unreachable destinations reasonably low for specific decoy deployments, and therefore argue RAD to be practical. Houmansadr et al. [22], however, show that re-routing of upstream traffic due to RAD is significantly costly to the censors, even if the ratio of unreachable destinations is kept small. This is intuitively because the re-routed traffic will discard the “best” BGP routes that

Table 1: Major decoy routing designs

System	Signaling	Upstream Channel	Downstream Channel	Architecture
Telex [59]	TLS ClientHello random nonce	TLS records for content	TLS records	Upstream (Type I)
Cirripede [21]	TCP ISN	TLS records	TLS records	Upstream (Type I)
Curveball [25]	Out-of-band	TLS records	TLS records	Upstream (Type I)
TapDance [58]	TLS ciphertext	TLS records	TLS records	Upstream (Type II)
Rebound [15]	Similar to Telex	TLS records using HTTP 404 error	TLS records using HTTP 404 error	Upstream (Type II)
Slitheen [4]	Similar to Telex	TLS records (HTTP header only)	TLS records (only image objects)	Upstream (Type II)
Waterfall	Registration	TLS records using HTTP 3xx redirection	TLS records	Downstream-Only

are meant to optimize performance and monetary expenses for the ISPs. Particularly, Houmansadr et al. [22] show that re-routed traffic due to RAD imposes the following classes of costs to the censoring ISPs:

- QoS degradation due to increased traffic latencies;
- QoS degradation due to increased route lengths;
- monetary expenses due to non-Valley-Free routing;
- monetary expenses due to switching traffic to less-preferred (more expensive) routes;
- monetary costs due to setting up new transit ASes;
- monetary and QoS costs due to massive changes in transit loads.

We refer the reader to Houmansadr et al. [22] for further discussion of the costs of RAD. More recently, Nasr et al. [41] performed a game theoretic analysis to optimize the placement of decoy routers against RAD censors. Earlier, others studied the problem of decoy deployment under non-adversarial threat models [10, 21, 29].

3 THREAT MODEL

Our threat model is the standard threat model considered in previous studies of decoy routing systems. We assume that the censoring ISPs use common censorship techniques including IP filtering, DNS interference, and deep-packet inspection to monitor and censor the Internet communications of their Internet users. The censors can also use various passive and active censorship techniques proposed in the literature (though not all of them have been witnessed in practice), such as statistical traffic analysis, active probing, and packet insertion/modification [18, 20, 54, 56, 57].

Like other decoy routing studies, we assume that the censors do not block encrypted communications (e.g., TLS) entirely. Doing so will impose significant collateral damage as encryption is essential to important business and entertainment Internet services. The censors, however, may selectively manipulate or disconnect encrypted connections. We also assume that the censors are not able to break the underlying foundations of TLS in order to man-in-the-middle arbitrary TLS connections (e.g., by compromising root certificate authorities or breaking cryptographic algorithms), as this will defeat not just decoy routing, but *any* privacy-enhancing technology.

We also assume that the censors are aware of the identities of the ASes deploying decoy routers and the locations of decoy routers. The censors can also use various BGP path inference tools to infer the BGP routes between any two points on the Internet.

We assume that the censors can re-route traffic inside their own ASes, advertise strategic BGP routes to their own ASes, and discard BGP routes to external ASes that contain decoy ASes. Similar to previous works on decoy routing, we assume that the censors are *not* willing and/or able to perform BGP poisoning attacks at large scale for a long time; we have elaborated on this in Section 4.3.

Finally, we assume that decoy ASes do not cooperate with the censors. ASes with strong business relationship with the censors, e.g., the ring (neighbor) ASes of China, will simply not run any decoy routers.

4 DOWNSTREAM-ONLY DECOY ROUTING

In this section, we introduce downstream-only decoy routing, which, as discussed above, offers significantly stronger resistance to RAD [22, 41, 48]. The core idea of our new architecture is to operate decoy routers *only* on the downstream traffic of the censored users, which is in contrast to previous designs. Intuitively, this makes our new approach stronger to routing attacks since a (censoring) ISP has significantly more control on the BGP routes of its upstream traffic, as opposed to its downstream traffic. Specifically, a censoring ISP can re-route its upstream traffic by applying per-destination rules (e.g., re-route only the traffic towards specific Internet destinations), however, it can *not* re-route downstream traffic by applying per-source rules (e.g., the censoring ISP has to either re-route all or none of its downstream traffic through each of its Internet provider ASes).

In the following, we will thoroughly evaluate the impact of rerouting attacks on downstream-only decoy routing through Internet-scale BGP path simulations. Our analysis demonstrates that our downstream-only decoy routing proposal offers significantly stronger resistance to the routing attacks by the censors compared to the traditional (upstream) decoy routing approach. Supported by our analysis, we conclude that *downstream-only decoy routing is a major step forward in making decoy routing systems practical*, as they need to be deployed on a significantly smaller number of volunteer ASes for a target resistance to routing attacks. For instance, we show that a downstream-only decoy routing system implemented on only a *single* decoy AS is as resistant to routing attacks (by the Chinese censors) as a traditional decoy routing system (e.g., Telex) with 53 decoy ASes.

Studied Routing Attacks. The original RAD attack studied in the literature [22, 41, 48] works by modifying upstream BGP routes, but *not* downstream routes. Therefore, it works against all previous decoy routing designs [4, 15, 21, 25, 58, 59] as all of them require their decoy routers to intercept upstream traffic of censored users. The original RAD attack, however, does not work on downstream-only decoys, as such decoy routers are oblivious to the upstream routes of the censored users (also note that the BGP protocol decides upstream and downstream routes between two end-points independently). We therefore introduce two variants of the RAD attack that are tailored to downstream-only decoy routing systems. We show that such attacks are extremely costlier to the censors compared to the original RAD attack on traditional systems, therefore concluding that downstream-only decoy routing systems offer superior resistance to re-routing attacks.

Simulation Setup. We use C-BGP [45], also used in prior studies [22, 41], to simulate Internet-scale BGP routing and evaluate the impact of different routing attacks using our Python code. We use the CAIDA's latest AS-relationship dataset [8], CAIDA's AS Rank dataset [9], and the MaxMind GeoIP dataset [37] to identify ASes of different countries and their relationships.

4.1 Source-Block Attack

In the original RAD attack, if the censoring ISP does not know any decoy-free upstream routes to a specific destination, she will block all upstream traffic to that particular destination. This makes the blocked destination "unreachable" to all censored users, but will prevent all censored users from using that route for (upstream) decoy routing. Our source-block attack works in a similar fashion on downstream traffic, as shown in Figure 2. More specifically, the censoring ISP entirely blocks traffic (e.g., by dropping packets) from an Internet AS *if* the BGP route *from* that Internet AS to the censoring ISP contains a decoy AS. Note that inferring downstream BGP paths [26] is not as accurate as inferring upstream BGP paths, however, we even assume that the censors can accurately infer downstream routes, which makes our analysis in the censor's favor.

4.1.1 Analysis. We simulate and evaluate the source-block attack on downstream-only decoy routing systems and compare its performance with RAD on upstream decoy routing designs. We consider China to be the censoring adversary, which is shown [22, 41, 48] to be the strongest routing adversary due to its well-connected Internet. We evaluate the attack for various numbers of decoy ASes. We choose decoy ASes based on an algorithm similar to what suggested in previous work [22, 41]: we simulate all routes from non-Chinese ASes to Chinese ASes, and pick transit ASes that appear more often on the routes to the censoring ISPs as the decoy ASes. We exclude Chinese ASes as well as the ring (i.e., neighboring) ASes of China from being decoys (since they have strong business relationships with China), as in previous work [22, 41, 48].

Figure 3 shows the unreachability costs imposed to Chinese censors due to the routing attacks. As can be seen, applying the source-block attack imposes a significantly larger damage to the Chinese censors compared to applying RAD on previous (upstream) designs. For instance, evading 100 upstream decoy ASes (e.g., in Telex) by the censors will disconnect Chinese users from only 8% of Internet destinations, however, evading 100 downstream-only decoy ASes will disconnect Chinese users from 23% of Internet destinations, therefore, imposing significantly larger collateral damage. In other words, a single downstream-only decoy AS causes the same unreachability damage to the censors as 53 upstream decoy ASes (therefore, for the same protection against routing attacks, downstream-only decoy routers needs to be implemented on a much smaller number of volunteer ASes).

4.2 Rewiring Attack

We also introduce and evaluate a more impactful (yet more costly) routing attack on downstream-only decoy routing systems, which we call *rewiring attack*. In this attack, a censoring ISP modifies the way it is connected to the Internet by disconnecting itself from some of its provider ASes and/or connecting to new ones, in order to reduce the number of decoyed routes available to its (censored) Internet users. More specifically, the censoring ISP will disconnect itself from an Internet provider AS that contains many decoyed downstream routes. The censoring ISP can even compensate the lost connectivity by connecting to new transit ASes whose downstream traffic contains fewer decoyed routes. An example rewiring attack is illustrated in Figure 4. As can be seen, the censoring ISP, AS1, has two links to the Internet through ring ASes (A) and (B). Suppose that the ring AS (A) is on a decoyed downstream route from some overt destination, however, assume that the ring AS (B) does not deliver any decoyed downstream routes. Therefore, the censoring AS disconnects herself from (A), e.g., by terminating their Internet transit contract, and will only use the AS (B) to connect to the Internet. In this case, the downstream traffic from the overt destination to AS1 will switch to a decoy-free route through ring (B). Note that this will re-route *all* downstream traffic previously received through (A) to (B), including the majority decoy-free routes.

The rewiring attack is practically unreasonable to the censors as changing Internet connectivity at large scale is not trivial. We yet evaluate this attack on downstream-only decoy routing to compare its impact with RAD on upstream decoy systems. Our analysis shows that even if the censoring ISPs undertake the irrational rewiring attack, it will cost them much more damage than what RAD does on upstream systems. The main intuition for such a higher cost is that a censoring ISP can not selectively re-route specific decoyed downstream flows, but she has to re-route all or none of the traffic through each of her ring ASes. On the other hand, the RAD attack can selectively re-route only upstream connections that transit trough decoy ASes. Additionally, even switching to another ring AS does not necessarily switch all downstream decoyed traffic to decoy-free routes depending on the placement of decoy routers, e.g., for overt destinations close to decoy ASes.

4.2.1 Analysis. We consider two types of rewiring attacks. In the first type, rewiring-I, a censoring ISP disconnects herself from some of its provider (ring) ASes without connecting to new AS providers. This will re-route traffic that the censoring AS received from its disconnected provider ASes to its other provider ASes. In the second type of the attack, rewiring-II, the censoring AS can even

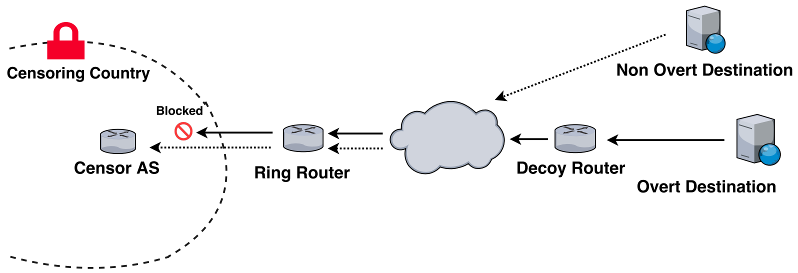


Figure 2: The Source-Block attack on a downstream-only decoy routing system. The censoring AS selectively blocks traffic from its neighbor AS A, by blocking downstream traffic that transits through at least one decoy AS (solid line) and allowing other traffic (dotted line).

Table 2: Comparing the impact of rewiring and RAD attacks on China for various numbers of decoy ASes. % Reroute and % Unreach are the fraction of rerouted and unreachable routes, respectively, and Impacted Routes is the sum of them.

	Rewiring attack on downstream-only systems					RAD attack on previous systems		
	Rewiring-I		Rewiring-II					
# Decoys	% Reroute	% Unreach	% Reroute	% Unreach	%Impacted Routes	% Reroute	% Unreach	%Impacted Routes
1	9%	13.0%	21%	0.6%	22%	0.3%	0.2%	0.5%
2	27%	13.1%	39%	1.3%	40%	0.9%	0.5%	1.4%
5	63%	14.6%	76%	1.6%	78%	2.5%	1.2%	3.7%
10	68%	15.6%	79%	3.4%	83%	5.4%	2.1%	7.5%
50	70%	22.4%	83%	10.3%	93%	12.6%	5.6%	18.2%
100	69%	26.6%	80%	16.9%	97%	17.1%	8%	25.1%

Table 3: Comparing the impact of rewiring and RAD attacks on Syrian censors for various numbers of decoy ASes. % Reroute and % Unreach are the fraction of rerouted and unreachable routes, respectively, and Impacted Routes is the sum of them.

	Rewiring-II attack on downstream-only systems			RAD attack on previous systems		
# Decoys	% Reroute	% Unreachability	% Impacted Routes	% Reroute	% Unreachability	% Impacted Routes
1	80%	1.1%	81%	0.9%	0.4%	1.5%
5	42%	42.9%	85%	18.9%	8.8%	27.7%
10	29%	58.4%	87%	26.1%	15.1%	41%

switch its re-routed traffic through other censoring ASes under the control of the censor (i.e., connect to new transit ASes).

Table 2 compares the impact of the rewiring attack on downstream-only decoy routing with the impact of the RAD attack on previous upstream decoy routing systems (China is the censoring adversary). As can be seen, for the same number of decoy routing ASes, the impact of the rewiring attack is significantly larger than RAD. For instance, with 50 ASes deploying decoy routing, *the RAD attack will impact only 18.2% of the traffic* of Chinese users (with 5.6% becoming unreachable and 12.6% re-routed), however, with the same number of decoys *the rewiring attack will impact 93% of China's routes*, i.e., 22.4% of routes become unreachable, and 70% get re-routed in the case of rewiring-I attack.

We also perform similar analysis for Syria to show the impact of the attack on a less-connected censoring country. As Table 3 shows, similar to China the rewiring attack imposes a significantly larger damage to the Syrian censors than RAD. For instance, with only 1 decoy router, *the rewiring attack will impact 80% of the routes as opposed to only 1.5% routes in the case of RAD*.

Importance of the costs As can be seen from our analysis, the rewiring attack has two types of consequences on the censors. First, it makes a large fraction of Internet destinations unreachable to the censors. Second, it enforces a large fraction of Internet routes to get re-routed to other Internet paths. *Both* of these are extremely unfavorable to the censors since they will impose significant monetary costs as well collateral damage to the censors, as shown in previous studies [22, 41, 48]. Particularly, Houmansadr et al. [22] have shown that the traffic re-routed by the censors is significantly more expensive and offer lower quality compared to the default (best) BGP routes, as also listed in Section 2.4.

4.3 BGP Poisoning Attacks

It is possible for a censoring ISP to use BGP poisoning attacks to modify the downstream routes of its users in order to avoid certain decoy ASes. One class of such attacks is the well-known BGP hijacking attack [3, 7] in which an adversary takes over the traffic

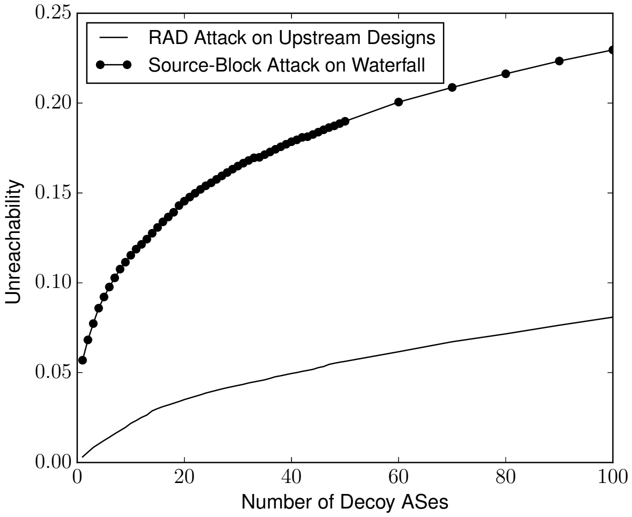


Figure 3: Comparing source-block attack on downstream-only decoy routing with RAD on previous designs.

to a victim AS by advertising fake shorter BGP routes or more-specific BGP routes for that AS. If a censoring ISP succeeds in BGP hijacking a decoy AS, she will be able to prevent censored decoy routing users (also non-decoy users) from reaching that AS for decoy routing. There are other known BGP poisoning attacks that may be leveraged by a censoring ISP. For instance, previous work [27] shows how carefully crafted BGP messages can route traffic away from a certain AS for the purpose of fast connectivity-failure recovery. Such tricks can be used by the censors to steer traffic away from decoy ASes. Consider a censoring ISP AS_C and a decoy AS AS_D. The censoring ISP can advertise the poisoning route AS_C-AS_D-AS_C to its neighbors. The neighbor ASes of AS_D will not advertise this route to AS_D if they have loop avoidance check enabled. In that case, AS_D will not know any paths to AS_C, and therefore, any traffic towards AS_C that would normally go through AS_D will then be re-routed through alternative routes (if there are any) or get dropped.

Note that such attacks are not specific to the decoy routing systems presented in this paper (downstream-only), but rather are relevant to any decoy routing system. Similar to previous studies, *we do not consider such attacks in our threat model* for the following reasons. First, to defeat decoy routing, a nation-state will need to deploy such BGP poisoning attacks *continuously* and *at large scale*. A rational nation-state is unlikely to do so due to the significant harm to its reputation. In fact, such attacks can be used for nation-state espionage and business competition, however, no nation-state is openly known to be performing them at large scale. Second, even if such poisoning attacks are implemented at large scale by a nation-state like China, Internet ASes can defeat it trivially as such attacks are easy to detect [7, 53, 60]. For instance, if a censoring ISP is known to hijack prefixes it does not own, other Internet ASes can implement additional inspection on the BGP messages from that ISP. Particularly, they can easily ignore any BGP advertisement that contains a loop to that censoring ISP [27] (i.e., a AS_C-AS_D-AS_C route). Note that *it is in the best interest of all Internet ASes to detect and defeat such BGP poisoning attacks*; continuous, large-scale BGP poisoning attacks severely harm the business interests of Internet ASes, e.g., by causing them to lose transit traffic routes due to poisoning attack, or by degrading the QoS of Internet traffic due to longer, less-stable routes. Some ASes (e.g., Cogent [27]) are already implementing mechanisms to defeat such attacks, e.g., by rejecting BGP advertisements from customer ASes if the advertised path contains one of their network peers or providers. Finally, note that unlike RAD and its alternatives presented in this paper, the BGP poisoning attacks are *not* inherent to decoy routing, but instead are due to weaknesses in the BGP protocol. Therefore, they may not apply to future, more-secure alternatives of the BGP like BGPSEC [34].

5 WATERFALL: THE FIRST DOWNSTREAM-ONLY DECOY DESIGN

As we demonstrated above in Section 4, the downstream-only decoy routing architecture offers a significantly stronger resistance to routing attacks compared to traditional decoy routing. This is enabled by running decoy routers only on the downstream traffic of the censored users, with no need to intercept the upstream traffic of the censored users. This, however, makes the design of downstream-only decoy routing systems *significantly more challenging* than traditional decoy routing systems. Particularly, all previous (upstream) decoy routing designs [4, 15, 21, 25, 58, 59] use the upstream traffic of the censored users for upstream covert communications, i.e., to communicate essential covert messages to the decoy routers, such as registration requests, HTTP GET requests for blocked destinations, etc. This is not possible in a downstream-only decoy routing system since the upstream traffic of the censored user is not supposed to be intercepted by the decoy routers.

In this paper, we demonstrate the feasibility of downstream-only decoy routing by designing *the first* downstream-only decoy routing system, which we call *Waterfall*. We use a set of complementary mechanisms to enable low-latency upstream covert communications in Waterfall despite its downstream-only architecture. Particularly, we use various HTTP redirection techniques to enable real-time upstream covert communications in Waterfall. We also use various techniques to make Waterfall resistant to traffic analysis attacks, as discussed in Section 9. We have built a *fully functional prototype* of Waterfall, which is available online [55].

5.1 Waterfall's Main Entities

The following are the key players in Waterfall, which is similar to any other decoy routing system.

Clients: A Waterfall client is a censored Internet user who installs Waterfall's circumvention software to bypass censorship.

Censor: A censor is a nation-state who regulates, monitors, and restricts the Internet access of its Internet users. A *censoring ISP* is an ISP under the jurisdiction of a censor, therefore one that implements the censorship mechanisms instructed by the censor.

Overt (Non-Blocked) Destinations: Internet destinations (e.g., webpages) that are not forbidden by the censor. Similar to other decoy routing systems, Waterfall's client software connects to some arbitrary overt destinations in order to bypass censorship.

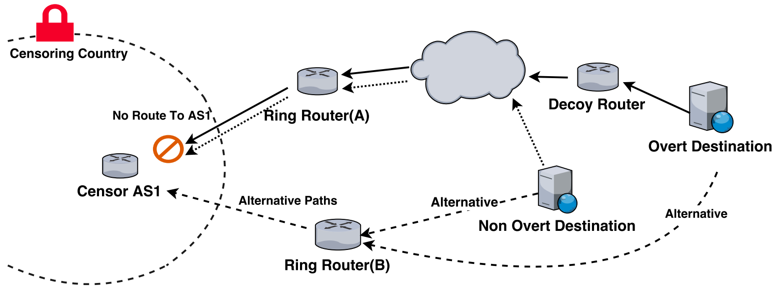


Figure 4: Rewiring attack on downstream-only decoy routing: The censoring ISP AS1 disconnects herself from the Internet provider (A), and re-routes all her routes through her other Internet provider (B).

Covert (Blocked) Destinations: Internet destinations that are forbidden by the censor, and therefore are not directly accessible by censored users. Censored users use Waterfall's client software to covertly communicate with blocked destinations.

Decoy AS: A friendly autonomous system (AS) who cooperates with Waterfall by mounting Waterfall's decoy routing software on its border routers, i.e., decoy routers. Previous work [41] has discussed the incentives of Internet ASes on becoming decoy ASes.

5.2 Overview of Waterfall's Operation

Figure 1c illustrates the main architecture of Waterfall. Here, we introduce the main phases of Waterfall's operation, which will be thoroughly discussed in the following sections.

Registration Phase To use Waterfall, a censored client first needs to register with the Waterfall system. During the registration, the client provides Waterfall with information required for the operation of Waterfall. Particularly, such information enables Waterfall decoys to authenticate the registered censored client, and to be able to man-in-the-middle the registered client's TLS traffic (with the client's consent). We will describe this phase in Section 6.

Circumvention Phase After the client has registered with Waterfall, she can use Waterfall to bypass censorship and connect to blocked Internet destinations, which is done in the following steps:

(1) *Establish an overt connection:* Like other decoy routing systems, a Waterfall client first establishes a TLS overt connection with an arbitrary non-blocked (overt) Internet destination. The overt destination should be chosen such that the downstream traffic to the client transits through a decoy AS (e.g., by trying random overt destinations until one is identified). Unlike previous designs, the upstream traffic to the overt destination does *not* need to transit through a decoy AS.

(2) *Authentication:* To serve registered Waterfall clients, Waterfall decoy routers need to identify the traffic belonging to the registered clients. This is done by using the information provided by the clients during their registration. Decoy routers also use client's registration information to authenticate Waterfall users, as well as to authenticate themselves to Waterfall users. This is required to protect the clients' confidentiality as well as to prevent manipulation of non-registered users' traffic. We will thoroughly describe this step in Section 6.

(3) *Covert communications:* Finally, an authenticated client covertly communicates with her intercepting decoy router through particular upstream and downstream covert channels. Through these channels, the decoy router proxies the client's traffic to the blocked Internet destinations requested by the client. We will thoroughly describe the design of Waterfall's upstream and downstream covert channels in Section 7.

We present the details of Waterfall's components below.

6 CLIENT REGISTRATION AND AUTHENTICATION IN WATERFALL

To be able to use Waterfall for circumvention, censored clients need to register with Waterfall's registration server. The registration server disseminates the registration information to Waterfall decoy routers for them to be able to authenticate and serve the registered clients.

There are two reasons why we need the client registration phase. First, registration enables a censored user interested in using Waterfall to inform Waterfall operators (e.g., decoy routers) of her interest. Waterfall decoy routers manipulate the traffic of only the registered clients, without interfering with the traffic of other Internet users who are not intending to use Waterfall. Even for each registered client, the registration phase enables the decoy routers to intercept only the specific overt connections intended by the clients. Second, the information exchanged during registration enables Waterfall decoy routers to man-in-the-middle the overt connections of the registered client (with their consents), which is required for circumvention. Note that most of the previous (upstream) decoy routing systems perform client registration (also called signaling) through a client's upstream traffic. This, however, is not possible in Waterfall since Waterfall decoys are not expected to be intercepting the client's upstream traffic.

Registration mechanism: Client registration is performed by a client sending a particularly formatted file, called the *registration package*, to a Waterfall registration server. Waterfall's registration phase is *not* latency-sensitive, therefore the registration package can be sent through any latency-insensitive communication channel. In our design of Waterfall, a client encrypts her registration package with the public key of Waterfall registration server, and emails the encrypted package to the publicly-advertised email address of the registration server. The server confirms the receipt of the package through a confirmation email.

Note that using email is not central to our design, as the encrypted registration package can be sent through *any* other latency-insensitive channel, e.g., by broadcasting on social networks or through domain fronted Tor bridges.

Registration package: A registration package is a file formatted as:

$$R_{Client} = Encrypt_{PK}(Client_{ID}|Conn^1_{ID}|Conn^2_{ID}|...) \qquad (1)$$

where PK is the public encryption key of Waterfall's registration server; $Client_{ID}$ and $Conn^i_{ID}$ are described in the following.

Client identifier ($Client_{ID}$). Client identifier is used by decoy routers to filter out unrelated overt traffic. A client with a static public IP address will use that IP as her ID. For a client behind dynamic NAT, the client will use her subnet as her ID.

Connection identifier ($Conn^i_{ID}$). Each connection identifier contains information that lets a decoy router identify an overt Waterfall connection created by a registered Waterfall client. Table 4 shows the format of a connection identifier. A connection identifier enables a decoy router to perform four main tasks:

(1) *Identify overt Waterfall connections:* A connection identifier allows a decoy router to identify the overt connection created by a registered client for decoy routing. In our design, the TCP ISN value is used for identifying such overt connections. Note that the TCP ISN is a 4 Bytes field that is generated randomly by the client establishing a connection. Waterfall clients will pre-generate such ISNs, and share them with the decoy routers during registration.

(2) *Man-in-the-middle the identified overt connections:* Waterfall decoy routers will only man-in-the-middle the overt connections that are identified as described above (i.e., by comparing the TCP ISNs). Figure 5 shows a typical TLS handshake. As can be seen from the figure, to be able to man-in-the-middle a TLS connection, a decoy router needs to know the client's TLS Random nonce as well as the client's public exchange key. Since the decoy router does not intercept the upstream traffic, the Waterfall client includes them as part of connection identifiers (Table 4). This will be discussed further in Section 7.

(3) *Authenticate Waterfall clients:* An adversary may send invalid registration information to Waterfall on behalf of a user who is not willing to use Waterfall (i.e., to DoS specific users or flood Waterfall). The adversary, however, has little chance in correctly guessing the values of the ISN and TLS Random nonces to be used by the non-Waterfall client. Therefore, once a decoy router succeeds in de-ciphering an overt connection's downstream traffic using a connection identifier provided in the registration package, the authenticity of the registration package is verified.

(4) *Secure covert communications:* The write and read keys (last column of Table 4) are used to encrypt the client's upstream and downstream covert communications with a decoy router.

Note that each registration package will contain multiple connection identifiers. To preserve unobservability, each connection identifier should be used only once for covert communications.

Size of the registration package: Each client identifier is 4 Bytes, and each connection identifier is at maximum 375 Bytes (as shown in Table 4). Therefore, a registration package with 1000 connection identifiers has a size of around 375 KBytes, which can easily be sent through a latency-insensitive channel like email. A client needs to send a new registration package to the registration server only after he runs out of unused connection identifiers.

7 COVERT COMMUNICATIONS IN WATERFALL

In this section, we describe how a registered client covertly communicates with the Waterfall decoy router intercepting her overt traffic. As discussed earlier, we use different channels for upstream and downstream covert communications in Waterfall. Figure 6 illustrates a typical covert communication in Waterfall. We assume that the decoy router has already authenticated the client based on her registration information, as described earlier in Section 6.

7.1 Upstream Covert Channel

A registered client uses Waterfall's upstream covert channel to send various messages to Waterfall decoys. Particularly, the channel is used by the clients to send to the decoys the upstream traffic (i.e., TCP packets) destined to censored destinations. The channel can also be used by the client to send various Waterfall-specific commands to a decoy router, such as commands to update registration information. Note that a client encrypts her upstream covert traffic using the `Write Keys` shared with Waterfall during the registration process (last column of Table 4).

To enable interactive, real-time browsing of the censored websites, the upstream channels between Waterfall clients and decoys should be low-latency channels. Previous designs [4, 15, 21, 25, 58, 59] (which are all upstream designs) establish this channel by simply embedding upstream covert messages into the upstream TLS records destined to an overt destination (whose upstream path is intercepted by some decoy routers). This, however, is not possible in a downstream-only decoy routing system like Waterfall since the decoys are not expected to be intercepting clients' upstream traffic to overt destinations. We therefore design several novel low-latency covert channels for upstream communications between Waterfall clients and decoys. Our channels offer lower capacity compared to the upstream channel used in previous (upstream) decoy proposals, however, the capacity suffices for upstream covert communications in Waterfall given the asymmetric nature of web traffic (i.e., upstream HTTP traffic has much less volume than downstream).

7.1.1 HTTP-based Channels.

We leverage various features of the HTTP protocol to establish several kinds of upstream covert channels in Waterfall, i.e., from a client to a decoy router on the downstream (but not upstream) path. The common feature of these HTTP-based channels is that they reflect (part of) the message being sent in an upstream HTTP message (which is *not* intercepted by downstream-only decoy routers) into a downstream HTTP message sent towards the client, therefore intercepted by Waterfall decoys. Figure 6a illustrates the main idea.

HTTP 404 Error. If a webserver receives an HTTP GET for a non-existing URL, the webserver will respond with an HTTP 404 error message; many webservers *include the invalid URL* in the returned error message. We leverage this to build an upstream covert channel for Waterfall. In order to send a covert message `CovertMessage`,

Table 4: The format of connection identifiers ($Conn^{i}_{ID}$), sent in a registration package.

Needed to identify connections	Needed to intercept connection	Needed to secure covert communication
TCP ISN #1 (4 Bytes)	Client public/private exchange keys (2 × 255 Bytes Maximum) Client TLS random nonce (4 Bytes)	Write Key, Write MAC Secret (56 Bytes) Read Key, Read MAC Secret (56 Bytes)
TCP ISN #2 (4 Bytes)	Client public/private exchange keys (2 × 255 Bytes Maximum) Client TLS random nonce (4 Bytes)	Write Key, Write MAC Secret (56 Bytes) Read Key, Read MAC Secret (56 Bytes)
⋮	⋮	

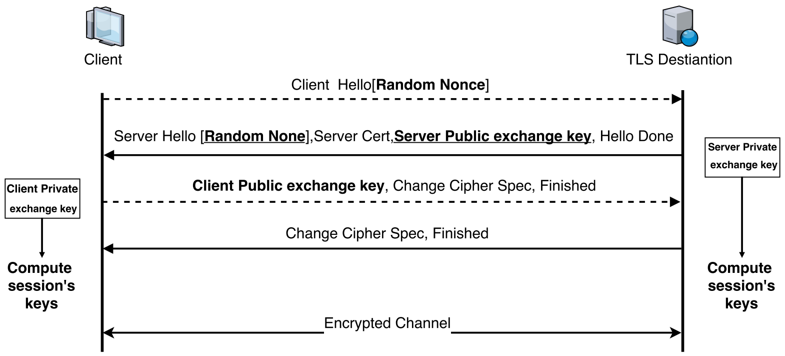
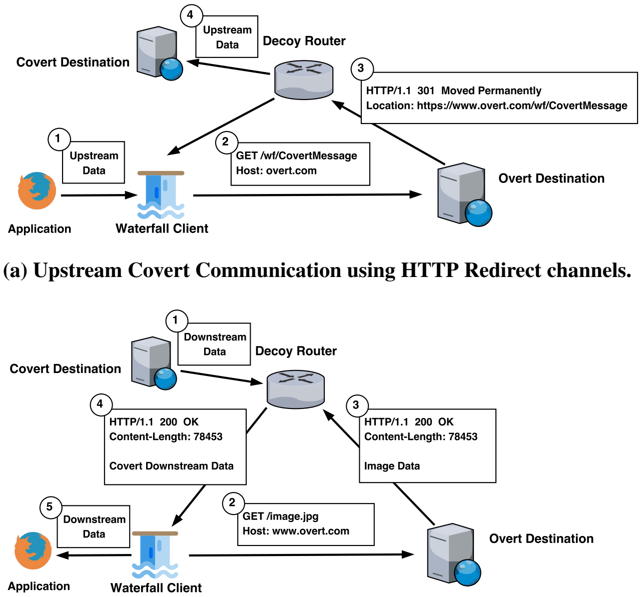

Client
TLS Destiantion
Client Hello[Random Nonce]
Server Hello [Random None],Server Cert,Server Public exchange key, Hello Done
Server Private exchange key
Client Private exchange key
Client Public exchange key, Change Cipher Spec, Finished
Change Cipher Spec, Finished
Compute session's keys
Compute session's keys
Encrypted Channel

Figure 5: A typical TLS handshake. The bolded elements are needed by a decoy router to intercept the TLS connection.



(a) Upstream Covert Communication using HTTP Redirect channels.

(b) Downstream Covert Communication

Figure 6: Covert communications in Waterfall.

a Waterfall client sends the following HTTP message to her overt destination:

```
GET /CovertMessage HTTP/1.1
Host: www.overt.com
```

Since /CovertMessage is not a valid URL, the server will respond the following back to the client, which will be intercepted by the decoy router on the downstream traffic:

```
HTTP/1.1 404 Not Found
Content-Length: 95

<html>
<title> Not Found </title>
The url <code> /CovertMessage </code> was
not found.
</html>
```

Note that not all websites replay the URL in the 404 error message. Particularly, our evaluation of the top 10,000 Alexa websites [2] shows that 3,916 websites support HTTPS, out of which 812 (20%) replay the URL in the 404 error.

HTTP 3XX Redirects. Typical Internet websites usually have multiple alternative hostnames. The HTTP 3XX redirection messages are designed to redirect a client to a different location. The most common use case of 3XX redirects is when a websites redirects to its `www` prefixed hostname, e.g. redirecting a request for `example.com` to `www.example.com`, or vice versa. What enables us to use HTTP 3XX Redirects for upstream covert communications is that such HTTP 3XX Redirect messages typically contain the whole requested path as part of the `Location` header. Therefore, a Waterfall client can send the following upstream HTTP message to her overt destination in order to send `CovertMessage` to a decoy:

```
GET /CovertMessage HTTP/1.1
Host: overt.com
```

The `overt.com` server will respond the following downstream message to the client, which will be intercepted by a Waterfall decoy router:

```
HTTP/1.1 301 Moved Permanently
Location: www.overt.com/CovertMessage
```

Note that `CovertMessage` can be any arbitrary covert message, for instance, `HTTP GET covert.com`.

Our evaluation of the top 10,000 Alexa websites [2] show that 50% (1,976 out of all 3,916) of all HTTPS websites perform HTTP 3XX Redirect to or from their `www` prefixed hostnames, therefore can be used for our upstream covert channel.

Channels tailored to specific (popular) webpages. Aside from the channels mentioned above, which work for a large number of overt websites, we also design upstream covert channels that are tailored to specific (popular) overt websites. Such channels exist because some websites reflect parts of the HTTP requests they receive from clients, such as the requested path or an HTTP header, back to the clients. In particular, sending the following request to the Google search engine:

```
GET /url?rct=j&url=CovertMessage HTTP/1.1
Host: www.google.com
```

will result in an HTTP response with the following `META` tag:

```
<META ... content="0;URL='CovertMessage'">
```

Similar to Google search, most websites with search capabilities tend to repeat the search query in the response HTML. Embedding covert messages in the search query will allow the decoy router to obtain the messages from the responses. Websites supporting this type of channel include popular search engines such as `google.com`, `bing.com`, `yandex.com` and `yahoo.com` and other websites with search capabilities such as `github.com` and `amazon.com`. Note that some (but not all) searchable websites inspect uncommon queries from users, e.g., by asking the user to solve a CAPTCHA.

7.1.2 Other possible channels. There are other possibilities to design low-capacity upstream covert channels for Waterfall. One direction is the use of known network steganography mechanisms [1, 40], such as timing channels, IPID, and TCP ISN. Such features will be reflected in the downstream traffic, and therefore intercepted by downstream decoy routers. These mechanisms, however, offer much lower capacities than the channels we introduced above. Another possibility is the use of heartbeat messages in the TLS protocol. The purpose of heartbeat messages is to make sure a connection is alive. A heartbeat message consists of a payload field that can be filled with arbitrary content [51]. A webserver receiving a heartbeat message will return the exact message on the downstream traffic, therefore it can be used as a Waterfall upstream covert channel. Note that, however, the use of TLS heartbeat messages is not very common in regular TLS connections, and therefore excessive use of it by Waterfall can raise suspicion.

7.2 Downstream Covert Channel

Designing Waterfall's downstream covert channel is more straightforward than its upstream channel, since Waterfall decoy routers intercept downstream traffic of the censored clients. We implement Waterfall's downstream covert channel by replacing the downstream TLS records of a client's overt traffic, as shown in Figure 6b. The decoy router is able to manipulate the downstream overt TLS records by using the information provided by the client in her registration package (Section 6). The decoys encrypt the downstream covert content using the `Read Keys` provided by the user during registration.

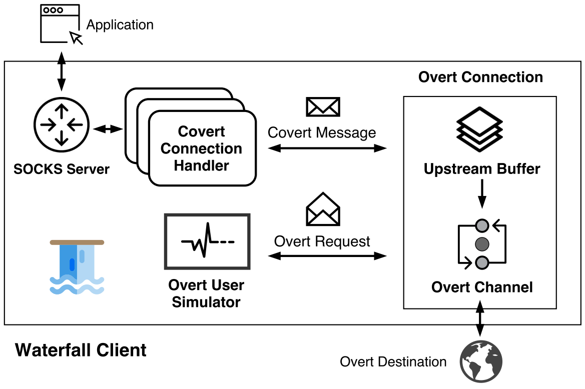


Figure 7: The architecture of Waterfall client software

Note that simply replacing overt TLS records with covert content will make Waterfall's traffic prone to various traffic analysis attacks, i.e., based on the timing, sizes, and order of packets. In Section 9, we describe how we carefully embed the upstream and downstream covert messages to resist traffic analysis attacks.

8 WATERFALL'S IMPLEMENTATION

We have implemented Waterfall as a fully operational system, which is available online [55].

8.1 Main components

Figure 7 shows the high-level architecture of the Waterfall client software, which is composed of the following components.

Application. This is *any* application software with SOCKS support that intends to use Waterfall to bypass censorship. We particularly use a web browser as the application component in order to make Waterfall for censorship-resistant web browsing.

SOCKS Proxy Server. A client application (e.g., a web browser) tunnels its traffic through Waterfall by connecting to a SOCKS proxy server run by Waterfall client software.

Covert Connection Handler. Each SOCKS connection made by a client application (e.g., an HTTP request by a web browser) is handled by a separate Covert Connection Handler (`CCH`). A `CCH` exchanges messages with the `OvertConnection` component, described below.

Overt User Simulator. The Overt User Simulator (`OUS`) is essentially a headless browser that creates overt traffic for Waterfall communications. The `OUS` does so by repeatedly sending requests for specific overt websites set by the client. As we describe in Section 9, the traffic generated by the `OUS` is used to resist traffic analysis attacks.

Overt Connection. The `OvertConnection` component embeds the covert data received from `CCH`s into the overt traffic generated by the `OUS`. More specifically, `OvertConnection` buffers the covert traffic received from `CCH`s until there is sufficient overt traffic that can carry the buffered covert traffic in a way resistant to traffic analysis. The `OvertConnection` decides an upstream covert mechanism

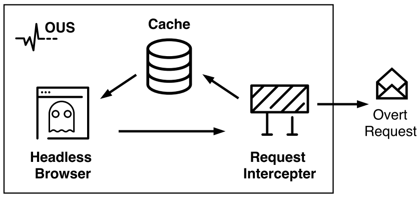


Figure 8: Waterfall's Overt User Simulator (OUS)

(from the set of mechanisms introduced in Section 7.1) for various overt destinations.

8.2 Step-By-Step Operation

We have described the step-by-step operation of our implemented Waterfall software in Appendix A.

8.3 Implementation Details

Our Waterfall client software implementation is written in about 900 lines of Python. It uses the Twisted Framework [52] for network communications and the Scapy [47] library for creating and manipulating TLS connections and messages. The `OUS` uses the Python Selenium library [49] to run and control a PhantomJS [44] browser. In order to intercept the overt requests, the PhantomJS browser is configured to use an HTTPS proxy running in the client which man-in-the-middles the overt requests using a self-signed certificate that the PhantomJS browser is configured to ignore.

Our Waterfall client supports the HTTP Redirect and HTTP 404 upstream covert channels (Section 7.1). It is possible to easily extend our client to support any other HTTP-based channels, including website specific channels, by simply providing a Python class that implements a given interface. A sample class for using Google as an overt destination with HTTP Redirects is shown in Appendix B.

We have also implemented a proof-of-concept decoy router using an experimental setup by simulating a network router intercepting downstream flows. We use `iptables` with `NFQUEUE` to intercept packets coming from overt destinations and process them with our decoy router code (~500 lines of Python). The decoy router code uses the negotiated client credentials (Section 6) to man-in-the-middle a client's TLS connection with an overt destination, and to extract covert upstream messages.

9 RESISTING TRAFFIC ANALYSIS ATTACKS

We describe how our implementation of Waterfall, described in Section 8, enforces mechanisms to resist traffic analysis attacks. To resist traffic analysis attacks, a decoy routing system must preserve the traffic patterns, i.e., packet timing and sizes, of the overt webpages in both upstream and downstream communications. Slitheen [4] is the state-of-the-art decoy routing system whose main objective is strong resistance to traffic analysis. To achieve unobservability, Slitheen embeds the upstream covert data within custom headers in the overt HTTP requests made by an OUS. To make room for covert data while preserving the original request size, Slitheen removes or compresses redundant HTTP header fields. Slitheen also preserves downstream traffic patterns by replacing data in HTTP responses of the overt website with covert data of the same size. However, in order to allow the OUS to complete a normal page load of the overt website, Slitheen only replaces responses containing leaf content (e.g., images and videos). Consequently, Slitheen is only able to use 40% of the available downstream throughput [4].

Similar to Slitheen our OUS aims at preserving traffic patterns in both upstream and downstream overt traffic while increasing downstream throughput compared to Slitheen. Note that we can not borrow the mechanism used in Slitheen (i.e., embedding covert data in overt request headers made by the OUS) since Waterfall decoys do not intercept upstream overt traffic. In order to send upstream covert data, Waterfall must make separate requests, e.g., requests that will result in HTTP 404 or 3XX Redirect responses. This complicates the task of imitating a normal user browsing the overt website.

The main intuition behind our OUS is to cache previously browsed overt websites, and only use already cached requests for upstream and downstream covert communications. This allows Waterfall to modify and replace requests to *perfectly mimic*[2] the traffic patterns of the overt website. On the other hands, this approach significantly increases the downstream covert capacity compared to the state-of-the-art Slitheen as Waterfall decoys are not limited to replacing only the leaf content. Figure 8 shows the block diagram of our OUS, composed of three main components. `HeadlessBrowser` is a background browser that repeatedly loads pages from overt websites. The OUS `Cache` stores unmodified responses to all overt requests previously received by the browser. Finally, `RequestInterceptor` intercepts the requests made by the `HeadlessBrowser`, responds to the browser with content from the `Cache`, and forwards overt requests to be processed by the overt connection in a traffic analysis resilient manner. Specifically, Waterfall client modifies `HeadlessBrowser`'s traffic based on the following criteria to preserve traffic patterns:

A. Non-Overt Requests: These are requests that are not directly targeted at the overt destination server, such as images hosted on CDNs or advertisements. Such requests are proxied normally with no modifications.

B. Cache-Miss Requests: These are requests whose responses do not exist in the `Cache`, e.g., they have not been previously requested or their TTLs have expired. Such requests are proxied without any changes to the traffic. The responses received in response to these requests are used to populate the `Cache`.

C. Small-Response Cached Requests: These are request that are cached in the OUS, where the size of the cached response is small, i.e., comparable in size to that of typical HTTP Redirect responses (we use a threshold of 1 KB based on our measurement of HTTP redirects in top 10,000 Alexa webpages). Waterfall replaces such requests with requests used for upstream covert communication with the decoy routers (e.g., through HTTP Redirect) as described earlier. To do so, Waterfall will create an overt HTTP request containing covert data, while preserving the original request size. Note that requests made by a browser typically have many HTTP headers, some with large values such as the `Cookies` header. We make room for covert data by omitting some of the non-essential header content. The request will be sent to the overt destination in place of the original request. In order to maintain a normal overt page load in the

[2] Note that since Waterfall runs the actual HTTP protocol, it is not subject to active/proactive attacks on imitation systems [20].

browser, upon receiving the response to this request, the OUS will respond to the browser's request from the Cache.

D. Large-Response Cached Requests: These are requests that are cached in OUS, where the size of their cached responses are larger than typical HTTP redirects or errors (i.e., 1 KB). Waterfall uses such requests for downstream covert communication as they offer ample downstream covert capacity. That is, the Waterfall client does *not* modify their upstream HTTP requests, however, the intercepting Waterfall decoy router replaces their (large) downstream responses with downstream covert messages, as described earlier. The decoy router preserves the size and timing of downstream responses in order to ensure resistance to traffic analysis.

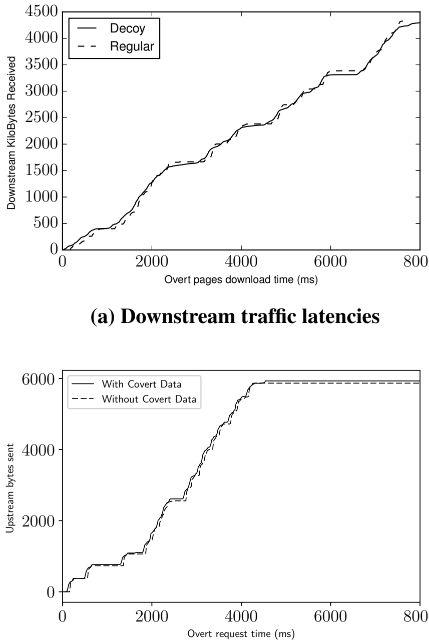
9.1 Evaluating Traffic Analysis Resistance

Our described implementation of Waterfall provides strong resistance to traffic analysis. This is because Waterfall's OUS and decoy routers preserve the exact timings and sizes of upstream and downstream overt traffic based on the traffic patterns cached from previous connections. We confirm resistance to traffic analysis through experiments on our implemented Waterfall. Using Google as the overt destination, we simulated a browser page load by making requests for different resources (e.g., HTML, JS, and images) from Google servers with a sequence of predefined delays across four seconds. The requests were made by a Python script configured to use OUS's `RequestInterceptor` as an HTTPS proxy. Figures 9a and 9b compare the CDF of the regular overt traffic with Waterfall traffic for downstream and upstream traffic, respectively (averaged over 30 runs). The figures demonstrate that Waterfall strongly preservers the traffic patterns of both upstream and downstream traffic, as described before. We confirm this statistically using a two-sided K-S test [36] on total download times. Our test results in a D-value of 0.11 and a p value of 0.5, showing that the K-S test fails in distinguishing between regular and Waterfall connections to an overt website with significant confidence.

9.2 Evaluating Throughput

Unlike the state-of-the-art Slitheen, which only replaces leaf content in downstream traffic, in Waterfall decoy routers can use the whole content of downstream TLS packets for covert communications. This is thanks to Waterfall's OUS preserving traffic patterns based on cached observations, enabling the client browser to continue loading the page regardless of the response received from the overt destination. As a result, Waterfall achieves a significantly higher downstream throughput compared to the recent Slitheen. Particularly, while a Waterfall decoy router uses 100% of the downstream traffic throughput for covert communications, Slitheen can only use 40% of downstream throughput (based on Slitheen [4]'s evaluations for top 1,000 Alexa websites).

As described earlier, a Waterfall client only uses "small-response" overt requests for its upstream communications. Therefore, the upstream throughput of Waterfall depends on the ratio of small-response requests contained in a full page load of the selected overt websites. We evaluated 1,976 websites that support HTTP redirects, and find that, on average, 40% of their upstream traffic consists of small-response requests, which can be used for upstream covert communications in Waterfall.



(a) Downstream traffic latencies

(b) Upstream traffic latencies

Figure 9: Traffic analysis resistance evaluation

10 CONCLUSIONS

We proposed a new architecture for decoy routing, called downstream-only, that is significantly stronger to rerouting attacks than *all* previous designs, as we demonstrated through extensive Internet-scale BGP simulations. We designed and built the first downstream-only decoy routing system, called Waterfall, leveraging novel covert channels. In addition to its strong resistance to routing attacks, Waterfall is designed to defeat traffic analysis. We believe that downstream-only decoy routing is a major step towards making decoy routing systems practical by significantly reducing the number of volunteer ASes needed to deploy decoy routers.

ACKNOWLEDGMENTS

We would like to thank anonymous reviewers for their feedback. This work was supported by the NSF CAREER grant CNS-1553301.

REFERENCES

[1] Kamran Ahsan and Deepa Kundur. 2002. Practical data hiding in TCP/IP. In *Proc. Workshop on Multimedia Security at ACM Multimedia*, Vol. 2.

[2] Alexa Top 1 Million Websites, February 2017. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip.

[3] Hitesh Ballani, Paul Francis, and Xinyang Zhang. 2007. A study of prefix hijacking and interception in the Internet. In *ACM SIGCOMM CCR*, Vol. 37. ACM, 265–276.

[4] Cecylia Bocovich and Ian Goldberg. 2016. Slitheen: Perfectly Imitated Decoy Routing through Traffic Replacement. In *ACM CCS*. ACM, 1702–1714.

[5] Justin Boyan. 1997. The Anonymizer: Protecting User Privacy on the Web. *Computer-Mediated Communication Magazine* 4, 9 (Sept. 1997).

[6] Bridge Easily Detected by GFW. https://trac.torproject.org/projects/tor/ticket/4185.

[7] Kevin Butler, Toni R Farley, Patrick McDaniel, and Jennifer Rexford. 2010. A survey of BGP security issues and solutions. *Proc. IEEE* 98, 1 (2010), 100–122.

[8] The CAIDA UCSD [AS Relationships] - [Jan 2016]. http://www.caida.org/data/as-relationships/.

[9] CAIDA AS Ranks. http://as-rank.caida.org.

[10] Jacopo Cesareo, Josh Karlin, Jennifer Rexford, and Michael Schapira. 2012. *Optimizing the placement of implicit proxies*. Technical Report.

[11] China blocks VPN services that let Internet users get around censorship. http://www.scmp.com/news/china/article/1689961/china-blocks-vpn-services-let-internet-users-get-\around-censorship. Online Article.

[12] Roger Dingledine and Nick Mathewson. Design of a Blocking-Resistant Anonymity System. https://svn.torproject.org/svn/projects/design-paper/blocking.html.

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

[14] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. 2013. Protocol misidentification made easy with format-transforming encryption. In *ACM CCS*. ACM, 61–72.

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

[16] Roya Ensafi, Philipp Winter, Abdullah Mueen, and Jedidiah R Crandall. 2015. Analyzing the Great Firewall of China over space and time. *PoPETs* 2015, 1 (2015), 61–76.

[17] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. 2015. Blocking-resistant Communication through Domain Fronting. In *PETS*.

[18] John Geddes, Max Schuchard, and Nicholas Hopper. 2013. Cover your ACKs: Pitfalls of covert channel censorship circumvention. In *ACM CCS*. ACM, 361–372.

[19] John Holowczak and Amir Houmansadr. 2015. CacheBrowser: Bypassing Chinese Censorship without Proxies Using Cached Content. In *ACM CCS*.

[20] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. 2013. The Parrot is Dead: Observing Unobservable Network Communications. In *IEEE S&P*.

[21] Amir Houmansadr, Giang TK Nguyen, Matthew Caesar, and Nikita Borisov. 2011. Cirripede: Circumvention Infrastructure Using Router Redirection with Plausible Deniability. In *ACM CCS*.

[22] Amir Houmansadr, Edmund L Wong, and Vitaly Shmatikov. 2014. No Direction Home: The True Cost of Routing Around Decoys. In *NDSS*.

[23] Amir Houmansadr, Wenxuan Zhou, Matthew Caesar, and Nikita Borisov. 2017. SWEET: Serving the Web by Exploiting Email Tunnels. *IEEE/ACM Transactions on Networking* 25, 3 (June 2017), 1517–1527.

[24] Jeffrey Jia and Patrick Smith. Psiphon: Analysis and Estimation. http://www.cdf.toronto.edu/~csc494h/reports/2004-fall/psiphon_ae.html.

[25] Josh Karlin, Daniel Ellard, Alden W Jackson, Christine E Jones, Greg Lauer, David P Mankins, and W Timothy Strayer. 2011. Decoy routing: Toward unblockable internet communication. In *USENIX FOCI*.

[26] Ethan Katz-Bassett, Harsha V Madhyastha, Vijay Kumar Adhikari, Colin Scott, Justine Sherry, Peter Van Wesep, Thomas E Anderson, and Arvind Krishnamurthy. 2010. Reverse traceroute. In *NSDI*, Vol. 10. 219–234.

[27] Ethan Katz-Bassett, Colin Scott, David R Choffnes, Ítalo Cunha, Vytautas Valancius, Nick Feamster, Harsha V Madhyastha, Thomas Anderson, and Arvind Krishnamurthy. 2012. LIFEGUARD: Practical repair of persistent route failures. *ACM SIGCOMM Computer Communication Review* 42, 4 (2012), 395–406.

[28] Sheharbano Khattak, Laurent Simon, and Steven J Murdoch. 2014. Systemization of Pluggable Transports for Censorship Resistance. *arXiv preprint arXiv:1412.7448* (2014).

[29] Donghyun Kim, Glenn R Frye, Sung-Sik Kwon, Hyung Jae Chang, and Alade O Tokuta. 2013. On combinatoric approach to circumvent internet censorship using decoy routers. In *MILCOM*.

[30] Lantern. https://getlantern.org/.

[31] Christopher S. Leberknight, Mung Chiang, and Felix Ming Fai Wong. 2012. A Taxonomy of Censors and Anti-Censors Part II: Anti-Censorship Technologies. *Int. J. E-Polit.* 3, 4 (Oct. 2012), 20–35. https://doi.org/10.4018/jep.2012100102

[32] Colin Lecher. How Iran Censors The Internet. http://www.popsci.com/technology/article/2013-03/how-iran-censors-internet-infographic. Online Article.

[33] Colin Lecher. Internet censorship reaching dangerous levels in Turkey. http://www.todayszaman.com/national_internet-censorship-reaching-dangerous-levels-in-turkey_393727.html. Online Article.

[34] Matthew Lepinski. 2015. BGPSEC protocol specification. (2015).

[35] Zhen Ling, Junzhou Luo, Wei Yu, Ming Yang, and Xinwen Fu. 2012. Extensive analysis and large-scale empirical evaluation of Tor bridge discovery. In *INFOCOM*. IEEE, 2381–2389.

[36] Frank J Massey Jr. 1951. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association* 46, 253 (1951), 68–78.

[37] MaxMind GeoIP Databases & Services: Industry Leading IP Intelligence. http://www.maxmind.com.

[38] meek Pluggable Transport. https://trac.torproject.org/projects/tor/wiki/doc/meek.

[39] Summary of meek's costs, July 2016. https://lists.torproject.org/pipermail/tor-project/2016-August/000690.html.

[40] Steven J Murdoch and Stephen Lewis. 2005. Embedding covert channels into TCP/IP. In *International Workshop on Information Hiding*. Springer, 247–261.

[41] Milad Nasr and Amir Houmansadr. 2016. GAME OF DECOYS: Optimal Decoy Routing Through Game Theory. In *ACM CCS*. 1727–1738.

[42] Daiyuu Nobori and Yasushi Shinjo. 2014. VPN Gate: A Volunteer-Organized Public VPN Relay System with Blocking Resistance for Bypassing Government Censorship Firewalls.. In *NSDI*. 229–241.

[43] Vasile C Perta, Marco V Barbera, Gareth Tyson, Hamed Haddadi, and Alessandro Mei. 2015. A glance through the VPN looking glass: IPv6 leakage and DNS hijacking in commercial VPN clients. *PoPETs* 2015, 1 (2015), 77–91.

[44] PhantomJS, scripted headless browser. http://phantomjs.org/.

[45] Bruno Quoitin and Steve Uhlig. 2005. Modeling the routing of an autonomous system with C-BGP. *Network, IEEE* 19, 6 (2005), 12–19.

[46] Phil Sands. Syria Tightens Control over Internet. http://www.thenational.ae/news/world/middle-east/syria-tightens-control-over-internet. Online Article.

[47] Scapy, Python Packet Manipulation Library. https://github.com/secdev/scapy.

[48] Max Schuchard, John Geddes, Christopher Thompson, and Nicholas Hopper. 2012. Routing around decoys. In *ACM CCS*.

[49] Selenium, browser automator. http://www.seleniumhq.org/.

[50] Michael Carl Tschantz, Sadia Afroz, Vern Paxson, et al. 2016. SoK: Towards Grounding Censorship Circumvention in Empiricism. In *IEEE S&P*. 914–933.

[51] Michael Tuexen, Robin Seggelmann, and Michael Williams. 2012. Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension. *Transport* (2012).

[52] Twisted, Python Networking Framework. https://twistedmatrix.com/trac/.

[53] Tao Wan and Paul C Van Oorschot. 2006. Analysis of BGP prefix origins during Google's May 2005 outage. In *IPDPS*. IEEE, 8–pp.

[54] Liang Wang, Kevin P Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. 2015. Seeing through network-protocol obfuscation. In *ACM CCS*. 57–69.

[55] Waterfall's Implementation. https://github.com/ha-D/waterfall.

[56] Tim Wilde. Knock Knock Knockin' on Bridges' Doors. https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors.

[57] Philipp Winter and Stefan Lindskog. 2012. How the Great Firewall of China Is Blocking Tor. In *FOCI*.

[58] Eric Wustrow, Colleen M Swanson, and J Alex Halderman. 2014. TapDance: End-to-middle anticensorship without flow blocking. In *USENIX Security*.

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

[60] Zheng Zhang, Ying Zhang, Y Charlie Hu, and Z Morley Mao. 2007. Practical defenses against BGP prefix hijacking. In *ACM CoNEXT*. ACM, 3.

[61] Hadi Zolfaghari and Amir Houmansadr. 2016. Practical Censorship Evasion Leveraging Content Delivery Networks. In *ACM CCS*.

A STEP-BY-STEP OPERATION OF OUR WATERFALL IMPLEMENTATION

In Section 8.1 we overviewed the main components of our implementation of Waterfall. To better illustrate the details of our implementation we will go through the steps taken for a browser to access a given censored website, `covert.com`, using the non-censored website `overt.com` as the overt destination.

Waterfall Startup

(1) The client establishes a TLS connection to `overt.com` using a pre-negotiated nonce value.

(2) The HTTP Redirect channel is selected to be used for sending upstream covert data for this overt destination.

Establishing a Tunnel

(3) The browser connects to the SOCKS server requesting to create a tunnel to `covert.com`. A new `CCH` (Covert Connection Handler) is created for this connection.

(4) The `CCH` creates a new covert message, `cm1`, which contains a command to establish a new tunnel with `covert.com`. It forwards the message to the overt connection component.

(5) The overt connection buffers the covert message in the upstream buffer.

(6) The OUS sends an HTTP request for the homepage of overt.com.

(7) The overt channel receives the overt request and observes that there is data available in the upstream buffer. It reads the message cm1 from the buffer and replaces the overt request with a request to https://overt.com/<cm1>. It then sends the new HTTP request on the overt connection.

(8) The overt website's server responds with an HTTP 301 message asking the user to redirect to www.overt.com/<cm1>.

(9) The decoy router intercepts and observes a covert message in the HTTP response. It reads the message and creates a tunnel to covert.com as instructed. The decoy router also assigns an identifier to the tunnel, which the client may use later to specify which tunnel it wishes to send data on.

(10) Once the client receives the tunnel's identifier, the CCH notifies the browser that the tunnel has been successfully created and it may start sending upstream data.

Sending Covert Upstream Data

(11) The user makes a request for the covert.com homepage in the browser.

(12) The CCH receives the TCP data for this request. It creates a covert message cm2 containing the request data and the identifier of the tunnel. It forwards the message to the Overt Connection that buffers the message in the upstream buffer.

(13) The OUS sends an HTTP request for an object in overt.com. The overt channel replaces the request with a request to overt.com/<cm2>.

(14) The overt website responds with a redirect allowing again for the decoy router to receive the covert message cm2. The decoy router reads the upstream TCP data from the message and forwards it on the specified tunnel to covert.com.

Receiving Downstream Data

(15) The covert website receives the client's request through the tunnel and sends the response back to the decoy router. The decoy router buffers the response.

(16) The OUS sends a request for an image in overt.com and the overt channel forwards the request on the overt connection without any modification.

(17) The overt destination responds with an HTTP response containing the image data. The decoy router intercepts this response and replaces the image data with buffered downstream data received from covert.com.

(18) The client receives the HTTP response, but instead of containing image data it contains covert downstream data. The CCH sends this downstream data to the browser over the SOCKS connection.

B SAMPLE WEBSITE-SPECIFIC OVERT CHANNEL IMPLEMENTATION

The following is an example covert channel class implementation for using Google as the overt destination with HTTP Redirects. The make_message method is used by the Waterfall client to create a request containing covert data which will be responded with a HTTP 301 Redirect from the Google server. The extract_message is used by the decoy router to extract the covert data embedded in the HTTP Redirect response received from the overt website.

```
class GoogleRedirectChannel:
  host = "google.com"
  prefix = "/waterfall"

  def make_message(self, overt_req, covert_buf):
    length = self.calc_covert_length(overt_req)
    data = covert_buf.read(length)
    encoded_data = base64.b64encode(data)

    request = Request()
    request.set_path(self.prefix + encoded_data)
    request.set_header("Host", self.host)

    return request

  def extract_message(self, request):
    loc = request.get_header("Location")
    reg = re.search("%s/(.+)" % self.prefix, loc)
    data = reg.group(1)
    return data
```

C POTENTIAL QUESTIONS

Here we present some of the useful questions raised during the reviwing process.

What if the censors disrupt Waterfall's registration by blocking Waterfall email messages? First, previous work [23] has studied censorship resistant emails that can be leveraged by Waterfall. Second, as discussed earlier, Waterfall registration does not fundamentally depend on emails, and any latency-insensitive channel can be used for its registration (one can even deploy multiple registration mechanisms in parallel). Given the small size of registration packages, Waterfall operators can even set up a resilient Domain Fronting [17] registration server at very low operational costs.

Can censors disable Waterfall by normalizing TCP ISN fields? We believe no! First, the use of ISNs only helps the decoy routers to filter out non-Waterfall traffic to reduce their traffic loads. Even if we eliminate ISNs from its design, Waterfall will still work: decoy routers will have to check every connection of a registered user to identify and serve Waterfall connections, as opposed to doing so only for flows the with specific, registered ISNs. Second, we believe that normalizing ISNs at large scale (e.g., by China) is not practical. As the upstream and downstream flows of a connection may take asymmetric routes, censoring ASes will need to share the ISN mappings for different flows among themselves in real-time.

Does a Waterfall client need to frequently probe for proper overt destinations? Would not this affect unobservability? Previous work [21] shows that the number of probes needed to identify such overt destinations is reasonably small. Also, given that routes do not change frequently, identified overt destinations can be used for long intervals. The Waterfall client software can also include a BGP path simulator to infer potential overt destinations, therefore minimizing the number of such probes. Nonetheless, this is not specific to Waterfall and applies to all decoy routing systems.