Piyush Kumar Sharma*, Devashish Gosain, Himanshu Sagar, Chaitanya Kumar, Aneesh Dogra, Vinayak Naik, H.B. Acharya, and Sambuddho Chakravarty

SiegeBreaker: An SDN Based Practical Decoy Routing System

Abstract: *Decoy Routing (DR)*, a promising approach to censorship circumvention, uses routers (rather than end hosts) as proxy servers. Users of censored networks, who wish to use DR, send specially crafted packets, nominally addressed to an uncensored website. Once safely out of the censored network, the packets encounter a special router (the Decoy Router) which identifies them using a secret handshake, and proxies them to their true destination (a censored site).

However, DR has implementation problems: it is infeasible to reprogram routers for the complex operations required. Existing DR solutions fall back on using commodity servers as a Decoy Router. But as servers are not efficient at routing, most web applications show poor performance when accessed over DR. A further concern is that the Decoy Router has to inspect *all* flows in order to identify the ones that need DR. This may itself be a breach of privacy for other users (who neither require DR nor want to be monitored).

In this paper, we present a novel DR system, *SiegeBreaker* (SB), which solves the aforementioned problems using an SDN-based architecture. Previous proposals involve a *single unit* which performs all major operations (inspecting all flows, identifying the DR requests *and* proxying them). In contrast, SB distributes the tasks for DR among three independent modules. (1) The SDN controller *identifies* DR requests via a covert, privacy preserving scheme, and does not need to inspect all flows. (2) The reconfigurable SDN switch intercepts packets, and forwards them to a secret proxy efficiently. (3) The secret proxy server *proxies* the client's traffic to the censored site. Our modular, lightweight design achieves performance comparable to direct TCP downloads, for both in-lab setups, and Internet based tests involving commercial SDN switches.

Keywords: Decoy Routing, Anti-Censorship, SDN

DOI 10.2478/popets-2020-0051
Received 2019-11-30; revised 2020-03-15; accepted 2020-03-16.

***Corresponding Author: Piyush Kumar Sharma:** Indraprastha Institute of Information Technology (IIIT) Delhi, India E-mail: piyushs@iiitd.ac.in

1 Introduction

Free speech on the Internet is a political battleground. On one hand, several governments claim that their sovereignty extends to cyberspace, and that they should control the content seen (or written) by their citizens. On the other hand, as censorship is frequently used to silence the opposition [13], access to free speech online is considered a human right by the United Nations [54]. This tension between censors and free speech advocates has led to an "arms race": users use proxy based services (*e.g.* VPNs and Tor [19]) to access content on the Internet, and in response, censors design stronger counter measures [22]. Even the best anti-censorship solutions, such as relay addresses [11] and protocol obfuscation [44, 57], are temporary measures [33].

Decoy Routing [34, 39, 61] is an attempt to break out of this "arms race", through the use of "smart routers" that double as proxies. DR clients – residents of censor countries, who need to access a blocked site – start by sending packets addressed to allowed sites (the *overt destination (OD)*). These packets *appear* innocuous, but actually carry a covert cryptographic message, carefully chosen to make it hard for the censor to distinguish the packets from regular TLS messages. Once safely past the network boundaries of the censor, these packets are identified by DRs that hijack the client's connection and redirect that flow to the *covert destination (CD)*, the actual censored site the client wishes to contact. Even

Devashish Gosain: IIIT Delhi, India E-mail: devashishg@iiitd.ac.in
Himanshu Sagar: IIIT Delhi, India E-mail: himanshu14046@iiitd.ac.in
Chaitanya Kumar: IBM Research, Singapore, E-mail: chaitanya@sg.ibm.com
Aneesh Dogra: IIIT Delhi, India E-mail: aneesh13014@iiitd.ac.in
Vinayak Naik: BITS Pilani, Goa, India E-mail: vinayak@goa.bits-pilani.ac.in
H.B. Acharya: RIT, USA, E-mail: acharya@mail.rit.edu
Sambuddho Chakravarty: IIIT Delhi, India E-mail: sambuddho@iiitd.ac.in

if a DR is discovered, it is hard for the adversary to "Route Around Decoys" [29, 35] without losing connectivity from a major part of the Internet.

Interest in Decoy Routing has led to a rich body of prior proposals [14, 21, 34, 46, 60, 61], but *not* led to the development of practical and efficient DR systems. There are several challenges that must be addressed before DR becomes widely adopted in practice.

Firstly, Internet applications perform poorly when using existing DR systems. As reported in previous research, DR clients experienced overall low throughput [26] and high latency [21]. This is because existing solutions are implemented on commodity servers (rather than real routers); they cannot serve traffic at line rates while at the same time matching network flows using cryptographic operations. On the other hand, it is virtually impossible to simply port existing solutions to conventional routers (which are not designed to be reprogrammed).

Secondly, existing proposals assume (unfounded) mutual trust between third-party DR operators (analogous to Tor maintainers), and the "friendly" ISPs (who simply *host* the DR). These proposals involve DR operators inspecting *all* flows traversing the ISP. Thus, DR operators may compromise the privacy of non-DR flows passing through "friendly" ISPs, (*e.g.* by recording their packet headers, metadata, or even content), breaching the trust extended to them.

A third issue is identifying the networks where DRs may be positioned (to intercept requests from several users), and providing incentives for these network operators to deploy DR. Finally, a fourth issue is defending against various attacks (traffic analysis *etc.*) from different adversaries.

The DR community has begun research efforts to address the third [29, 35] and the fourth issue [14]. However, the first two problems – assuring *performance* for regular DR flows, without compromising *privacy* of non-DR flows – remain open challenges to practical DR.

In this paper, we propose the first practical system to answer such concerns: *SiegeBreaker (SB)*, a Software-Defined Network (SDN) [42] based, efficient, and privacy preserving DR system. It works as follows:

- The users of SB signal the Decoy Router through an email, addressed to an email ID associated with the SDN controller (that manages the SDN switches). The email bears encrypted information, that can only be decrypted by the SDN controller; at a later step, this information authenticates a legitimate DR client to the controller.
- After deciphering a DR request, the controller reconfigures the switch (*i.e.* the Decoy Router), to redirect all subsequent TCP packets, *of only the SB client – OD connection*, to itself.
- These TCP packets carry covert information (indicated in the previous email message) that authenticate a legitimate DR request, *without* requiring the inspection of all other flows traversing the DR (as required in all previous proposals). This drastically reduces the inspection load on the controller.
- After authenticating the client, the controller reconfigures the switch to divert subsequent SB client – OD packets (*i.e.* only the DR flows) to a third-party *secret proxy (SP)*.
- These subsequent packets help covertly establish a fresh session key with the SP, and expose to it the URL of the CD. The SP then communicates with the CD and sends the responses back to the client, ensuring reliable and efficient delivery of packets.

Overall, the protocol ensures that the third-party DR operator managing SP, observes only the DR requests, without having any knowledge of the non-DR flows, thereby preserving their privacy.

SB proves to be a practical and privacy-preserving DR system. It builds upon the idea that DR is not a monolithic task. Thus, it distributes the tasks of DR over three loosely-coupled, synergistic modules: (1) The SDN *controller*, that orchestrates the switch(es), independently identifies DR requests and re-configures SDN switches on-the-fly. (2) The *switch*, which primarily forwards packets between the client and the SP, *seldom* engaging the controller. (3) The *SP*, which focuses on communicating with CD, along with other tasks to ensure secrecy and reliability. Such a design not only leads to a massive improvement in performance, but also ensures that the DR system (and its operator) only sees those few flows which are associated with DR requests. Our major contributions can be summarized as follows:

1. The prototype of SB, a practical and efficient DR solution, which achieves multiple goals:
   - Modularity. DR requests are handled by three loosely coupled entities – the SDN controller, the switch and the SP – reducing computational load on each.
   - Privacy preserving signalling. Using the above modular design, SB inspects a small fraction of all traffic and serves *only* the DR requests.
   - Dedicated hardware. SB uses SDN switches (instead of commodity servers proposed in previous designs) to divert flows efficiently at line rates.

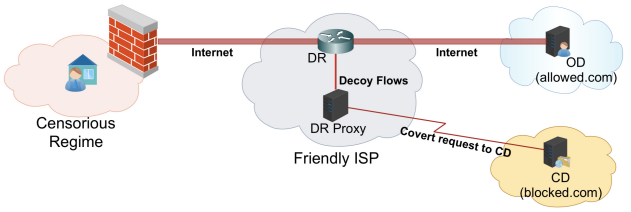

Fig. 1. Decoy Routing: The user access `blocked.com`, through DR. To the censor it appears that the client is communicating to an unfiltered site, `allowed.com`.

  - Reliable communication. SP provides reliable and efficient client–CD communication, by preserving the characteristics of TCP like behaviour (when there is no actual TCP connection between them).
2. Comprehensive performance analysis of SB, on real testbeds (involving a commercial SDN switch), shows near-native (TCP) performance for *both* web surfing and bulk downloads, even in the presence of 50K parallel flows or 15Gbps network-traffic on the SDN switch. In ideal conditions, SB is capable of serving individual client requests at line rates of $\approx$ 1 Gbps.
3. Thorough security analysis of the protocol, showing how SB is resilient to a variety of advanced attacks (such as forced asymmetry [51]).

Finally, we note that existing efforts to deploy DRs [34, 35] could be very costly — even modest proposals [29] involve replacing about 11,000 routers across 30 ASes with DRs. However, in an SDN-based AS, *any* switch could double as a DR on-demand, without disrupting regular routing. This solves the problem of DR placement *within an AS*. We further demonstrate how the client could use inconspicuous protocol messages to covertly signal the controller, and help identify the best switch for installing flow rules, in Subsec. 4.4.

2 Background

2.1 Decoy Routing

DR is an anti-censorship mechanism that that uses "friendly" routers in the Internet as proxy servers. Being routers, they are very difficult to block (ref. Fig. 1). The steps for DR are as follows: **(1)** A user hosted in a censor regime sets up a TLS connection with an unfiltered website (the OD), hosted outside the boundary of the regime. **(2)** The client-OD traffic carries a special cryptographic signature which can be identified by DRs en-route. On identifying this signature, the DR hijacks the connection, and engages a proxy (the DR proxy), that finally fetches the content from the CD. **(3)** The traffic from CD is returned to the client by the SP, setting TCP/IP header fields to appear as if they were part of the initial client–OD TLS connection.

2.2 Software Defined Networking (SDN)

Software Defined Networking [23] is an emergent network design paradigm that decouples the control and data plane of the network. An SDN architecture has a centralized controller, which controls the operation of the switches in the network (usually, by sending commands over a standard protocol named OpenFlow [6]).

In SDNs, complex operations (such as computing network-wide control policies) are the responsibility of the *controller*. The policies, in the form of *flow tables*, may be installed dynamically on the network switches which are simple dedicated devices, merely forwarding traffic based on these tables. The advantage of such an architecture is that control plane functionality is consolidated in a single, programmable entity – the controller.

Also, the SDN controller is capable of adding (or updating) any flow table rule or to redirect traffic to itself for analysis. By default OpenFlow compliant switches redirect any previously unseen packet, for which it has no matching rule, to the controller for inspection.

Since SB is also an SDN based anti-censorship system, it naturally inherits the salient features of SDN. In our design the controller and the SDN switches maintained by a friendly ISP are trusted; whereas the Secret Proxy managed by the DR operators/volunteers is not. Thus, any traffic analysed by the controller, will not compromise the ISP's regular customers' privacy, as it is maintained by the ISP itself.

2.3 Related Work

Existing DR systems follow the general pattern described in the previous section, but vary in their choice of secret handshake, side channel, and whether they take additional measures to suppress unwanted messages from the OD to the client (which might make the censor suspicious). We now provide a brief survey of existing DR systems.

Karlin *et al.* [39] proposed an initial DR architecture, called Curveball. However, the first-generation DR implementations include Cirripede [34] and Telex [61]. Cirripede uses the TCP Initial Sequence Number (ISN) field as a covert channel for registering the client to the *Registration Server*. This server provides a fixed set of sequence numbers that the client may use in subsequent connections. In Telex, the client embeds a cryptographic tag (created using the Telex station's public key) in the

TLS nonce field of the Client Hello message to OD. Seeing a tag, the Telex station establishes a shared secret with the client (using Diffie-Hellman key exchange), which is subsequently used to covertly proxy traffic to the filtered sites. Telex requires *both* sides of the client–OD communication be intercepted; it is, thus, fragile where protocol messages (and acknowledgements) between the client and OD traverse asymmetric network paths. The recent systems by Bocovich [14, 15] and by Nasr [46] build upon the signaling architecture of Telex and Cirrepede respectively.

Tapdance [60], an improvement over Telex [61], uses chosen-ciphertext steganography to signal the DR in an incomplete HTTPS request destined to the OD. Further, this request eliminates the need for inline blocking of the OD [61] and the requirement to intercept both request and response, as the OD never responds to such requests. Recently, Frolov *et al.* [26] partnered with a small ISP (and a university) to deploy Tapdance for practical usage. Their experiments reveal that incomplete HTTPS requests, often terminate in roughly 30 seconds, requiring frequent re-negotiation. This issue, coupled with the average Tapdance client throughput of $\approx$ 5KBps, makes it unsuitable for accessing much of the web content. This reveals how existing DR solutions struggle with real-life deployment. Even on a very small ISP, with three uplink routers (compared to thousands, for a backbone ISP like Level-3 or Cogent [29]), there are serious performance problems. Deploying at scale would require a much larger infrastructure, capable of monitoring millions of flows, identifying the DR requests, and providing them proxy service.

Bocovich *et al.* [14] took a different approach where they address latency-based and website-fingerprinting attacks. They achieved resistance to such attacks by sending the covert content in the leaf elements (images *etc.*) of the webpage served by OD. The client maintained an active connection with OD, and only the leaf content in the response of the OD was replaced with covert content. Further, as Slitheen still required analysing content in both the reverse and forward path, the same authors in their subsequent paper [15] proposed *Gossip*, a new protocol which allows all existing routing symmetry-dependent DR systems to work despite path asymmetry. Another approach was proposed by Nasr *et al.* [46], in which the *downstream* content, *i.e.* replies from OD, are used to detect DR, rather than the (upstream) requests. This helps in the mitigation of strong RAD [51] attacks. However, both Slitheen and Waterfall were developed and deployed on commodity servers, which are *not practical* to scale in an ISP.

In brief, existing DR solutions have not succeeded in utilizing commodity routers to match/decrypt messages and serve as proxies on-demand at an ISP scale.

3 SiegeBreaker vs Prior Research

We now describe how SB significantly differs from prior attempts at DR (summarized in Tab. 1).

- ***Implementation On Real Routers:*** Existing DR proposals use commodity servers as Decoy Routers – either by using the server as a router [34], or by attaching the server to one (through a wiretap) [60]. A server running a non-realtime OS, that has to inspect *all* the network connections to identify DR requests, and also proxy such requests to the intended CDs, is a clear bottleneck. SB uses hardware (SDN) switches instead of commodity servers, allowing it to process ISP-scale traffic at line rates. Although, Tapdance was deployed in an ISP in 2017 [26], it did all the processing on a server attached to a router. It is important to note that, unlike SB, their implementation was not on the router *itself*.
- ***Real Internet Workload Performance:*** Among existing DR prototypes, Frolov [26] demonstrates the practical use of Tapdance, inside a university and within an ISP. However, they reported overall low client throughput (5KB/s), and precludes testing with large files. Other approaches have restricted themselves to downloads of a few (less than ten) small ($<$ 1 MB [34]) webpages, in a controlled laboratory environment [60, 61], and even so, report over half a minute to download the home pages of popular sites ($<$ 1.5 MB in size) [21].
  SB's prototype is *extensively* tested for large files (up to 1GB) and high client link bandwidths (up to 1Gbps). Even under heavy cross-traffic conditions, and with CD and OD over the Internet, SB's client requests can be served at line rates of 1 Gbps. SB's performance was comparable to direct TCP downloads (near-native) in *every* scenario we tested.
- ***Privacy Preservation of non-DR clients:*** All prior approaches require the DR to inspect both the DR and non-DR flows, either by analysing TLS or TCP SYN packets[1]. This allows the DR volunteer

1 In Cirripede and Waterfall, if we assume that the registration server is maintained by the ISP's operator, then they could inherit this property. Additionally, how SDN implementation can be used to complement existing DR schemes (with this feature) is elaborated in A.2.

Properties	Siege-Breaker	Curve-ball	Telex	Cirripede	Tap-Dance	Rebound	Slitheen	Waterfall of Liberty	Secure Asymmetry
Implementation on real routers	●	○	○	○	○	○	○	○	○
Real Internet Workload Performance	●	○	○	○	○	○	○	○	○
Privacy preservation of non-DR clients	●	○	○	●†	○	○	○	●†	○
Handling asymmetric routing	●	○	○	●	●	●	○	●	●
Resistant to replay attacks	●	●	●	●	○	●	●	●	●
Resistant to latency analysis	○	○	○	○	○	●	●	●	●
Resistant to website fingerprinting	○	○	○	○	○	○	●	●	●

Table 1. A comparison of different features of existing DR schemes. ● - feature supported; ○ - feature unsupported; ●† - feature supported with some assumptions.

hosts (*e.g.* Telex/Tapdance/Slitheen station) to also see what sites other non-DR clients are visiting. This may be considered an unwelcome intrusion on their privacy. We introduce a novel signaling scheme that allows the controller to easily isolate DR requests (*i.e.* re-configure the SDN switch to selectively forward DR packets to the SP). Not only is this efficient (reduces load on the system) but also *privacy-preserving*, *i.e.* removes the need to inspect non-DR flows.

– ***Handling Asymmetric Routing:*** Routing of traffic in the Internet is not always symmetric, and most DR systems need to intercept client–OD traffic in both directions [14, 39, 61]. Some systems survive route asymmetry through special schemes (such as a gossip protocol) [15]. However, a few DR systems [21, 60] have no trouble with asymmetric routes; SB is one of these robust systems.
– ***Reliability and Efficiency:*** In all DR systems, the DR *hijacks* client–OD TCP connections. DR traffic between the client and the CD lacks the reliability and flow-control guarantees of regular TCP connections, and is thus easily impacted by background Internet cross-traffic. SB emulates TCP's message transmission mechanism, to ensure reliable, in-order and efficient delivery of packets to the client. This has significant positive impact on clients' performance.
– ***Limitations of SiegeBreaker:*** SB is not resilient to latency based [51] and website fingerprinting attacks [31]. Moreover, we have not tested SB in an ISP. (Such testing has only been demonstrated for Tapdance.) One may ask that in an SDN-based ISP, on which switch should the redirection rules be installed? As a solution, we have proposed a novel scheme (in Subsec. 4.4) using which a client can covertly signal the controller about the appropriate switch. Our proposed scheme, however, may be susceptible to fingerprint attacks by a determined adversary. In such scenarios, the controller would install rules on all the switches. But, due to the associated timeout, these rules would be eventually purged out from all the switches, except the one that would actually intercept the DR flows. This is explained in detail in Subsec. 4.4. However, we acknowledge, it still increases overhead on the system in terms of installing the unwanted rules whenever a user requests DR service.

4 System Design

4.1 Threat Model

We consider two types of adversaries for our system. Our primary adversary is a powerful nation-state that censors its citizens' Internet access. Not only can adversary censor regular Internet traffic using any known technique (DNS injection, IP address/URL filtering, DPI, *etc.*), it may also covertly monitor network packets. Further, in an attempt to detect/disrupt circumvention systems alone, it may also subtly manipulate network packets that cross its boundaries, without disturbing regular Internet users. Also, for the functioning of SB, we assume that users have access to *some* form of regular email service (either webmail or SMTPS).

We also consider a secondary (Byzantine) adversary — a motivated individual or group, who intends to disrupt normal network routing. This adversary sends forged messages, aiming to install unwanted rules in SDN switches, or in any other way disrupt network routing through attacking the SB infrastructure.

4.2 SiegeBreaker Protocol

Pre-requisites

The SB architecture assumes a network of centrally controlled SDN switches, configured to function as regular routers (*i.e.* they forward packets based on their desti-

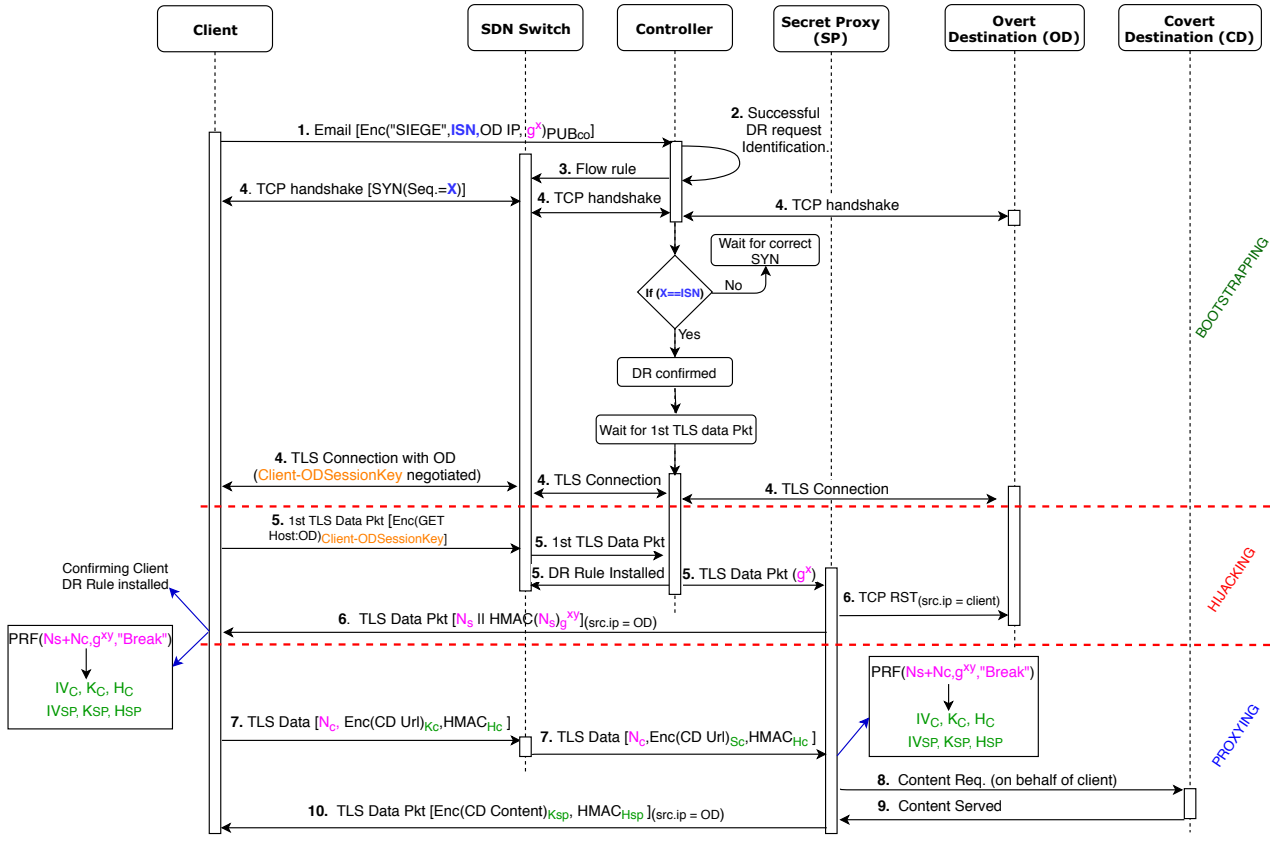

Fig. 2. SB Protocol: The numbered arrows correspond to the various protocol messages, exchanged between the client, the SDN switch (acting as DR), the OD, the SP, and CD, as described in Subsec. 4.2.

nation IP address). It is assumed that the client *a priori* has access to the following entities:

1. Public key (PUB_{CO})of the controller's 2048-bit RSA public-private key-pair.
2. The 256-bit DH public exponent g^y of SP.
3. The controller's email ID.

Protocol Description

SB's DR protocol comprises roughly of three phases—*viz. bootstrapping*, *hijacking* and *proxying*. We now present a step-by-step walk-through of a DR session, with reference to Fig. 2.

Bootstrapping (Phase I):

1. The client initiates DR by sending an email to the controller's email address. The payload of this email contains the DR request. It is encrypted with the public key of the controller, and thus not understandable to the client's email provider or the adversary.
2. The controller, on receiving an email, attempts to decrypt the content using its private key, and searches for four fields embedded in it: **(1)** The magic word "SIEGE", signifying DR request. **(2)** A TCP Initial Sequence Number (ISN) that the client will eventually use while handshaking with the OD. **(3)** IP addresses of client and OD **(4)** A DH exponent public value (g^x), the client derived using privately chosen number x.
3. On successfully identifying a DR request, the controller installs a rule on the SDN switch, so that *all* client–OD packets are redirected to it (the controller). This *inspection rule* will expire after a hard timeout.
4. Subsequently, the client initiates a fresh TCP handshake with the OD. The SYN packet of the handshake bears the *same* ISN, as the one sent in the initial email.
   This packet is redirected to the controller (as per the rule installed in step 3), who checks that its ISN matches the ISN specified in the initial email and forwards it to OD. Thereafter the client completes the TCP connection, and initiates a TLS handshake with the OD, negotiating the session key (Client-ODSessionKey).

This initial ISN match is the vital step that authenticates the client and identifies the DR requests, without the need to inspect any other flow.
In case the client's SYN packet does *not* have the expected ISN, the controller continues to poll the packets of the client–OD flow until the timeout period of inspection rule lapses, causing it to expire.

Hijacking (Phase II):

5. After completing the TLS handshake, the client sends a TLS data packet, carrying the `GET` request (with OD as the host field), encrypted with the session key that was negotiated with the OD.
   The inspection rule is still in place, so this packet goes to the controller which *replaces the payload* of the TLS data packet with the client's DH exponent g^x (which it received in step 2) and the OD IP. It then forwards this packet to the SP.
   Further, the controller also updates the redirection rule on the switch, to divert *all subsequent packets of the flow* to the SP. This rule matches the client-OD flow based on the client's source port, client IP and OD IP as seen in the TCP SYN packet. We denote this rule the *SP redirection rule*, and associate an idle timeout with it. Once a rule is installed for a specific client-OD pair, no other rule, for the same pair, would be installed until the said rule times out. Thus, clients behind a NAT firewall who try to use the DR service, would require selecting different ODs. This ensures that all such clients can simultaneously access the DR services. This is explained in detail in Subsec. 6.1.
6. The SP, on receiving the first TLS data packet, assumes that the client intends to use DR.
   SP terminates the existing client–OD connection by sending a RST packet (spoofed as client, with correct sequence and acknowledgement numbers) to the OD IP[2] (obtained in step 5).
   Further, the SP derives a pre-master key (PMK) (g^{xy}) using client's g^x (received via the first TLS data packet) and its own DH private number, y. The SP (spoofing as OD) then crafts a TLS data packet carrying a random nonce N_S, along with its HMAC computed using PMK.
   To the censor, this appears to be a regular TLS data packet, carrying random bits. The client, however, treats these bits as a nonce N_S and calculates its HMAC. It derives the PMK (g^{xy}) using the private DH number x, and the publicly known g^y. Successful verification of the HMAC allows the client to confirm that the DR request was successful.

Master-key derivation: Following a successful validation of the nonce N_S, the client selects its own nonce N_C. Using these nonces and the PMK, the client derives a 6-tuple key (IV_C,K_C, H_C, IV_{SP}, K_{SP}, H_{SP}). IV_C,K_C and H_C are the IV, cipher and HMAC keys for the client, while IV_{SP},K_{SP} and H_{SP} are those of the SP. These keys are generated using a Psuedo Random Function (PRF) involving SHA-256 hash function (Referring to TLS v1.2 [12]).

Proxying (Phase III):

7. Thereafter the client crafts a TLS data packet carrying N_C, the CD URL (encrypted with key K_C using 256-bit AES in CBC mode), and a HMAC of the URL and N_C (using HMAC key H_C).
   The client sends this packet, addressed to the OD. En-route the packet encounters the switch that redirects it to the SP.
8. SP, on successful reception of the aforementioned TLS data packet, extracts the nonce N_C. Using N_C, N_S and the PMK, SP also computes the same 6-tuple as the client. Upon successful HMAC validation, the SP decrypts the URL (using K_C). Finally, SP connects to the CD and requests data. The SP focuses entirely on serving DR requests, *without the burden of identifying them from among all flows via costly cryptography operations.*
9. CD serves the requested content to SP.
10. The SP encrypts these responses with key K_{SP} and signs them with HMAC key H_{SP}. It then sends them back to the client, spoofing the source IP address of the OD (maintaining the state of Client–OD connection). This keeps up the pretense, to the client's censor ISP, that the client is communicating with the OD. The same session, can also be used for requesting content from various other CDs (before the idle timeout expires).

This walk-through raises several questions, such as why email was used as a covert channel? Among multiple switches, exactly on which switch(es) the controller would install the inspection/redirection rules? What rule timeout values should be selected in our system? Given that there is no true TCP session between client and CD, how do they compensate for dropped packets *etc.* We now explain the design decision of using email

2 The RST causes the client–OD connection to be forcefully terminated, without any responses being sent to the client that may raise the censor's suspicion.

along with how SB would select switch(es) to install the rules and discuss the remaining concerns in Sec. 6.

4.3 Improved Covert Signalling

Existing DR implementations use TCP initial sequence numbers (ISNs) [34], the ClientRandom field inside TLS client hello [61], and the encrypted body of an HTTPS request [60] as covert signaling fields. These schemes require analyzing either *all* SYN packets [34, 46] or TLS flows [60, 61], as the covert patterns are embedded in the packets sent from the DR client to the OD. Inspecting large volume of traffic for identifying the embedded covert patterns, in innocuous looking packets, make it difficult for the censor to detect DR requests. However, these well thought signalling schemes pose new performance challenges for the DR users.

All existing DR systems rely on commodity servers to analyze the large volume of traffic [24]. This may pose as a performance bottleneck as NICs of these servers are not built to handle such high speed traffic. Further, these systems analyze all flows, including the non-DR ones, to detect DR requests. This may inadvertently compromise the privacy of non-DR users. In principle, SB can easily adopt any of the existing signalling schemes. However, SB aims to reduce the burden of analyzing all flows, while still retaining the same standard of unobservability (from the censor), compared to existing architectures. Additionally it also aims to preserve the privacy of non-DR users.

Thus, our covert signaling indicates the controller about the upcoming DR flows, such that it inspects *only the potential* DR traffic, reducing the amount of traffic to be analyzed. For this, a client can rely on any out-of-band channel (*e.g.* IMs, SMS and email *etc.*). In our present implementation, SB clients use an email to covertly signal the controller. This email contains client's source IP, OD's IP and the TCP ISN that would be used by the client in the subsequent DR request. Thereafter, the controller installs inspection rule for only flows with indicated client IP and OD IP address, *i.e.* a potential DR flow, *disregarding the rest of flows.* For potential DR flows, the controller matches the ISN in the SYN packet, with the one indicated in the email. A successful match, confirms a DR flow, and the SP redirection rule is installed on the switch.

Thereafter, only the DR flows are diverted to the SP, which has no knowledge of non-DR flows. This final step helps preserves the privacy of non-DR users from the DR volunteers maintaining the SP.

It must be noted that, as an alternative to ISN, the TLS ClientRandom may be sent in the email. The controller would thus confirm DR flows by matching the content of TLS ClientRandom in ClientHello (of potential DR flows), to the one received in the email.

On the use of email: In the past, email has been successfully used as a covert channel for transporting censored content [32, 36, 37, 41]. Similar to such efforts, we also assume that users have access to some form of TLS supported email (either webmail or regular SMTPS). However, in our design email is not directly used as the data channel; it is a *control* channel, used for signalling the SDN controller.

A determined adversary may attempt to snoop and block all emails destined to controller's mail ID. Since the communication is TLS encrypted, our approach is not vulnerable to wire sniffing adversaries. They are only vulnerable to a very powerful adversary who can assume control over the clients' email services. However, even when the email provider is controlled by the adversary, one can issue unique email IDs to each individual client, as proposed by Houmansadr *et al.* [37]. This makes it impractical for the censor to learn and block *all* such email IDs.

Moreover, an adversary may cripple the system by randomly delaying suspicious emails, thereby delaying the installation of the inspection rule. Oblivious DR clients' packets would thus reach the switch before this rule is installed. Hence, they would go undetected by the controller, failing to avail DR service. Further, our email body contains four fields and thus may have a fixed length, thereby raising suspicion. However, we make it difficult for the adversary to identify such emails. *E.g.* we pad the email body with random nonces.

However, a determined adversary may find some other identifiable patterns in the encrypted (registration) email. In such cases, we can resort to using other carriers, such as images, PDFs *etc.,* to stegonographically encode the covert signal. as already used in existing anti-censorship systems [16, 43]. However, our current implementation does not incorporate this feature.

In the extremity, an adversary may attempt to delay every email (both DR and non-DR). In such cases, SB may be tailored such that the controller sends an acknowledgment email back to the client. Reception of this email, confirms the availability of DR services to the client. As a drawback, this may incur a delay in the bootstrapping phase. However, in the presence of such disruptive adversaries, the email confirmation assures availability, albeit at cost of such delays.

4.4 Auxiliary Signalling Scheme

In SB, the controller needs to install `OpenFlow` rules to redirect the packets of a client–OD flow to itself or the SP. However, as a network will have multiple SDN switches, SB poses a new problem: how can the controller know *which* switch to install these rules on?

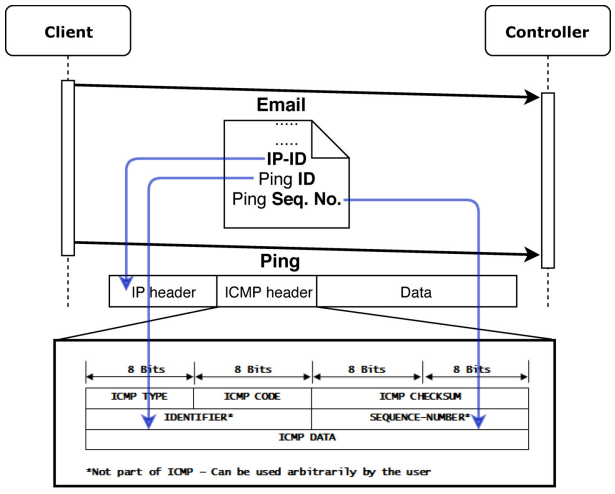

Fig. 3. `Ping` packet assisting the controller in selecting the appropriate switch to install redirection rule.

The simplest solution would be to flood *all* switches in the SDN with the redirection rules. But intuitively, this approach maybe expensive considering rule installation time and the finite memory of the switch. (Note that a rule is added for *every* client–OD flow. They cannot be aggregated.) We therefore propose a secondary signal to achieve the same. The client sends a crafted `ping` packet as a covert signal, to help the controller identify the switch it should install the DR rule on (ref. Fig. 3). The overall approach is explained as following:

1. Switches are bootstrapped with a rule that forwards all `ping` packets to the controller.
2. While crafting the email to the controller, the client adds three more fields: (i) IP-Identifier (from IP header), (ii) `ping` sequence number and (iii) `ping` identifier (from ICMP header).
3. Next, the client sends a crafted yet innocuous looking `ping` packet to the OD (with three fields described in previous step). En route to OD, decoy AS's SDN controller receives the `ping` packet and compares the aforementioned three fields to the one sent in the initial email. A successful match confirms that the `ping` packet is indeed from the authenticated client.
4. At this step, the controller selects the switch from which it received (and validated) the ping packet. Thus the redirection rules are installed on this switch, assuming the subsequent client-OD packets shall also arrive at the same switch.

Selection of three fields: The goal of the controller is to correctly associate the ping packet to the legitimate DR client that initially sent the email. An adversary may try to brute force this association by sending fake ping packets (spoofed as DR client), guessing the three header fields. In case the adversary succeeds, and the route taken by packets of the adversary differ from that of the client, the controller would end up installing a rule on a switch which might not appear on the client–OD route. This renders DR service unusable for intended DR clients. With our architecture, adversary needs to correctly identify 48 bits (IP-ID, ping seq. no. and ping ID, each being 16-bit field) making the brute force attack highly impractical (requiring $\approx$ 281 trillion `ping` packets per client–OD pair.)

Indistinguishable Pings: We only rely on the IP-ID, sequence number and identifier fields of the ping packet for covert signalling (which are by default random numbers). We keep rest of the fields (and payload) identical to *pings* generated by standard OSes. Thus our ping packet appear indistinguishable from regular ping packets generated by popular OSes.

However, we acknowledge that sending `pings` for covert signalling, before accessing DR, may be classified as a pattern (*i.e.* a fingerprint attack). We try to make it difficult for the adversary to detect such patterns by making these `ping` packets innocuous. Thereafter, we introduce random delays between `ping` packets and the corresponding DR requests.

Our auxiliary signalling mechanism is prone to fingerprint attacks. However, SB can also function without this scheme. In the absence of this scheme, the rule would be installed on all the switches. But, these would be automatically purged, after a short duration (due to timeouts) from all the switches, except the one which later identifies the actual SB clients packets. However, the installation of rules on all the switches may incur an additional memory overhead. This may eventually impact the total number of clients that can be supported by the switches. We thus analyzed the number of clients that can be supported simultaneously by the SDN switch. The `HP3500yl` SDN switch, that we used for evaluating our system (described ahead in Sec. 5), supports 74K table entries [3]. More advanced SDN switches [5] *e.g.* `HP10500` series can support $1,152,000$ table entries (*i.e.* $1,152,000$ openflow clients). Such switches can transport a total of about

3.8 Tbps [2] traffic. Moreover, the SDN switches that we used, do not impede performance when the number of clients (or entries in the table) increases. These are built for commercial deployment within large ISPs, and can easily handle a large volume of traffic without any slowdown. Thus, we believe that our proposed system will be suitable for deployment.

5 Experimental Evaluation

In this section, we describe the experiments conducted to evaluate the performance of SB. First, to test the efficacy of the protocol, we tested SB on DETER [1] testbed. We describe this in detail in Appendix A.1. Next, to comprehensively evaluate the performance, we tested SB on a physical testbed (using a commercial SDN switch, *viz.* `HP3500YL`). Our performance tests are broadly divided into two categories:

1. *Controlled experiments*, which were designed to test the *robustness* of the complete system including the client, proxy, and the SDN switch in a lab setup.
2. *Internet experiments*, which were designed to show that our system works well for *realistic scenarios* — *i.e.* downloading content from Internet websites that were otherwise blocked.

We begin by describing the setup used in the experiments, followed by the tests performed, and their respective results.

Experimental Topology:

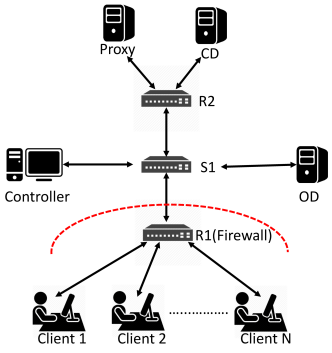

Fig. 4. The topology used for evaluating SB; 1 hardware SDN switch (`S1`), 2 routers (`R1`,`R2`), an SDN controller and 6 host nodes. R1 blocks traffic to CD.

SiegeBeaker was tested using the setup in Fig. 4, for controlled experiments (where all the entities were hosted inside our lab), and the setup in Fig. 8 for Internet experiments (when OD and CD were websites hosted over the Internet).

5.1 Controlled Experiments

Our first experiment involved the SB client downloading files of various sizes from the CD, and comparing the download times to that achieved when downloading via `wget`. The results of this test (ref. Fig. 5) depicts that SB performs considerably well in comparison to `wget`.

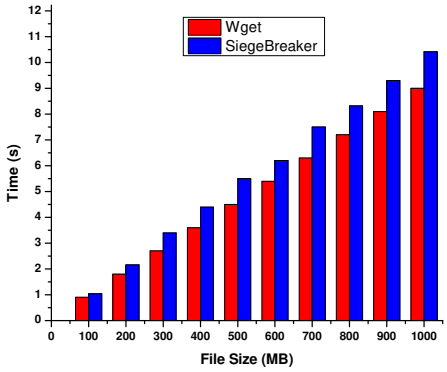

Fig. 5. Comparison of SB and `wget` on a controlled setup, in terms of download time for file sizes up to 1GB.

Our next experiment was to evaluate the performance of SB's SP–client congestion and flow control (explained in Sec. 6) against native web (TCP) traffic. Both SB and `wget` clients simultaneously downloaded large files (varying between 100 MB and 1GB) from different CDs, over 1 Gbps shared network link (between R1 and S1, see Fig. 4). The throughput achieved over this shared link, for both SB and `wget` is shown in Fig. 6. As evident from the results, SB and the `wget` client achieve roughly the same throughput (sharing the link capacity almost uniformly). The result demonstrates that our system *effectively emulates TCP's congestion and flow control* mechanisms, evenly sharing the link capacity with background TCP traffic.

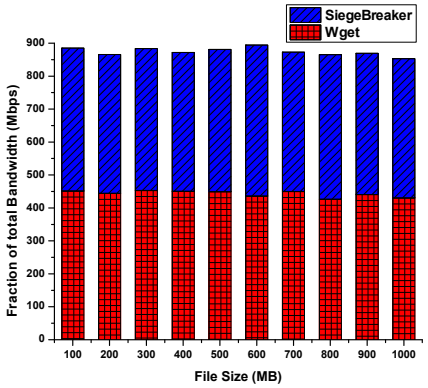

Fig. 6. TCP performance: SB and `wget` simultaneously downloading a file on a common link. They share the available bandwidth almost equally.

To determine the impact of cross-traffic on SB's performance (by increasing the load on SDN switch), we connected several hosts to the switch and exchanged increasing volume of traffic between them, via P2P connections. At the same time, we used a SB client to download a 1GB file from the CD. The results of this test are shown in Fig. 7. As evident, increasing the cross-traffic load had no impact on the client's throughput. This is because, once configured, the switch merely forwards selective DR flows to the SP. This effectively isolates them from the rest of the traffic, avoiding contention.

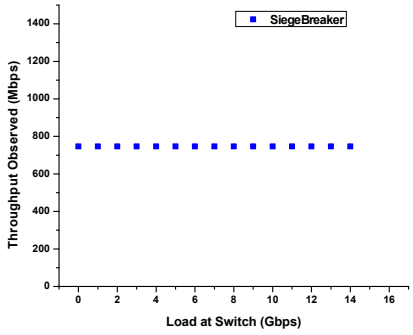

Fig. 7. SB client's performance with increasing load on the SDN switch.

Further, we also included provisions at the proxy to handle multiple clients. For a sample run of 9 clients simultaneously downloading a 100 MB file, we observed a nearly even distribution of bandwidth *i.e.* ≈ 11%. Similar results were observed when this experiment was repeated for larger files (such as 200 MB).

5.2 Internet Experiments

To further evaluate our prototype, we conducted experiments where the OD and CD were chosen to be websites hosted on the Internet. We tested our system's performance against two kinds of network workloads — regular web browsing and large file downloads respectively.

Assessing the performance for regular web browsing: This experiment involved recording the download times for the the home pages of several blocked websites. To simulate web censorship, we configured our university firewall to filter access to Alexa top-500 sites, for a particular client under our control. The SDN switch and the SP were installed outside the firewall, and had unhindered access to the Internet.

In our test we spawned 500 concurrent SB client instances, using blocked Alexa top-500 sites as CDs, and several random unfiltered sites as ODs. In all cases, we were successful in downloading the home pages of the

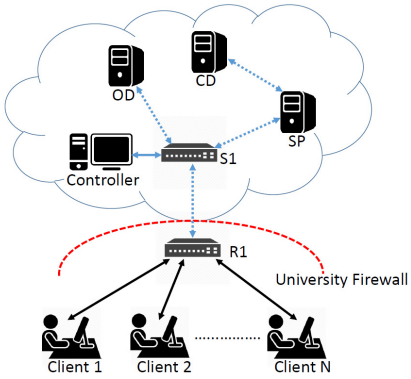

Fig. 8. The topology used for evaluating SB over Internet with clients inside a university campus.

blocked sites, which varied in size from a few kB to 1.5 MB; we measured their respective download times. Next, we reconfigured the firewall to disable filtering, and again accessed the same web pages using `wget`. As Fig. 9 shows, the average download time for SB (1.8 s) was very close to that for `wget` (1.7 s).

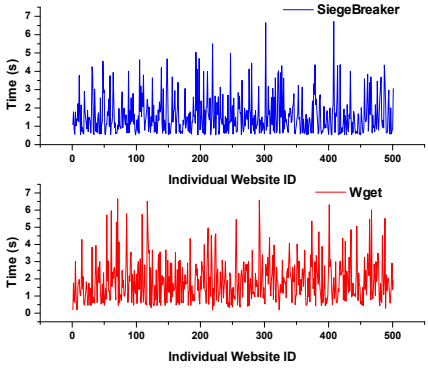

Fig. 9. Download times for Alexa top-500 websites — accessed by 500 parallel clients of SB and `wget`.

Assessing the performance for bulk downloads: To test SB with bulk file downloads, we set up web servers on six geographically distributed machines, each serving files of various sizes. The firewall was set to block direct access to these machines from our clients, but we could download the files using SB, using a random unfiltered website as OD. Regardless of the background network conditions, the download times for SB do not significantly differ from our baseline, *i.e.* download times using `wget`. Fig. 10 represents one such case, corresponding to a cloud server, comparing the download times *w.r.t* various file sizes (varying from 100 MB to 1 GB).

To gauge the performance of SB under heavy cross-traffic loads, we initially planned to divert the entire university's traffic through the switch. However, we chose against doing so, for two reasons. Firstly, it would force users uninvolved in the study to send their traffic

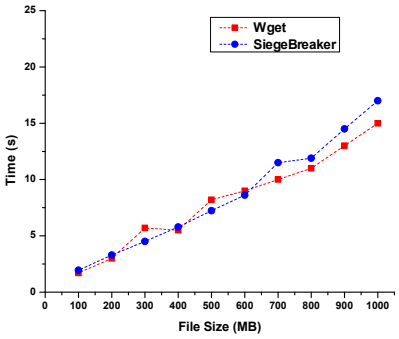

Fig. 10. Comparing time taken by SB and `wget` for downloading files, with OD and CD hosted on Internet.

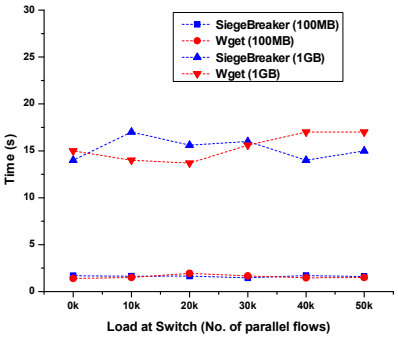

Fig. 12. SB vs `wget` when increasing the number of flows on SDN switch with OD and CD hosted on Internet.

through our switch, raising ethical concerns. Secondly, the cumulative volume of our university's traffic rarely exceeded 500 Mbps, and would not saturate the switch.

We therefore continued with our original setup, but connected 15 hosts to the switch so as to generate varying background loads. These hosts generated cross-traffic by establishing P2P connections with one another and exchanging large volumes of data ($\approx$ 15 Gbps). At the same time, we downloaded large files (100 MB and 1 GB) from the cloud servers, using both SB and `wget`. As Fig. 11 shows, SB's download times are comparable to those of `wget`, regardless of variations in cross-traffic.

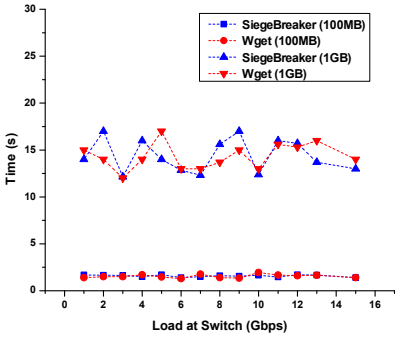

Fig. 11. SB vs `wget` when increasing the cross-traffic on SDN switch, with OD and CD hosted on Internet.

Similar tests were repeated by varying the number of parallel flows traversing the switch. The hosts, instead of exchanging P2P data, ran `Apache Benchmark Version 2.4` and connected to a web server (also connected to the switch), spawning as many as 50k concurrent web connections[3]. Fig. 12 presents the outcomes of these tests. Increasing the number of flows, had *no impact* on the download times achieved using SB. This was comparable to what is achieved using `wget`. Thus, even in the presence of high cross-traffic, SB proves to be practical for downloading files of all sizes.

Additionally, SB has no measurable impact on non-DR traffic due to increasing DR traffic. Non-DR traffic is forwarded directly by the SDN switch without any intervention from the controller. Since SDN switch is a specialized hardware designed to transmit traffic at line rates, the non-DR traffic is thus not impacted.

Overall, the promising performance of SB can be attributed to our modular design (which makes use of hardware SDN switches), and avoids the unnecessary cryptographic inspection of non-DR flows.

Measuring the setup time: A SB client is required to covertly signal the controller to install a DR redirection rule, before it begins the actual download from the CD. The time elapsed between initial email and the installation of final DR redirection rule (*i.e.* between step 1 and step 5 of protocol) is called *setup time*.

In 100 experimental trials, we observed an average setup time of 3-4 seconds. Similar to Sweet [37], we observed that most of this delay comes from email handling modules (selenium scripts composing mail, SMTP connection time at controller *etc.*) rather than network latency (which is of the order of milliseconds). [**Note:** The download time figures in previous subsections do *not* include setup time.]

Browsing experience: We used `wget` for benchmarking purposes. However, SB is also integrated in a browser. It can easily load static as well as dynamic websites without any significant performance degradation. Similar to Telex [61], we also used SB in our lab to access normal day to day websites for months without any problems.

5.3 Implementation Details

We now describe the implementation details of each individual component of SB. All in-lab machines, had In-

3 According to the NOC, this is *much* larger than the total number of flows at the university's edge router, that rarely exceeds 20k flows.

tel(R) Core(TM)i5-7400 CPUs @ 3.00GHz, provisioned with 8GB RAM, fitted with Gigabit Ethernet adapters.
Client: The client code is written in C (≈ 1100 LoC). It uses the `OpenSSL` library v1.1.0f for handling TLS connections. We use raw sockets API for crafting packets and `libpcap` for receiving and processing packets.

The email component is implemented for both SMTP and webmail using Python. The webmail version uses `Selenium` 3.14 [8] and SMTP version uses smtplib [10] in Python. The client program listens on a local port on the client machine for new websites requests and serves the content as and when requested.
Secret Proxy: The code for SP is also written in C (≈ 1500 LoC). The client – SP TLS connections are handled using the OpenSSL library. We modify the header fields via raw sockets and use `libpcap` for packet processing.
SDN switch: Our implementation uses the `HP3500yl`, a 24-port Gigabit SDN switch. This switch works in hybrid mode, *i.e.* both Openflow and non-Openflow environments simultaneously. It has three different types of immutable tables: *Default* (0), *Hardware* (100) and *Software* (200). Our redirection and inspection rules all make use of the Hardware table, as this allows for the best performance.
Controller: We designed and implemented our controller application using `Ryu` v4.15, which is written in Python (≈ 2068 LoC), and communicates with the switch using `Openflow` 1.3. Email reception and processing is implemented for both IMAP and webmail in Python. The webmail version uses Selenium and the IMAP version uses Python `imaplib` [4]. In our experiments we used IMAP for fetching Emails, as it was easier to integrate with `Ryu` controller.
OD and CD: We use Linux nodes running Apache v2.4.18 configured as HTTPS server, to serve as the OD and CD.
The source code of SB can be found at at [9].

6 Discussion

In this section, we discuss the various design aspects along with the security analysis of the SB protocol.

6.1 System Design

1) ***Selection of timeouts:*** In our design, the controller needs to install redirection rules at steps 3 and 5. As the switch has limited memory, these rules need to be purged eventually.

The inspection rule (step 3 of protocol) match packets based on client–OD IP pair and has a hard timeout associated with it. We cannot use an idle-timeout for it because if the DR and non-DR clients are behind a common NAT device (*i.e.* share same source IP), then the aforementioned rule would also capture packets for the non-DR clients accessing the same OD. The rule may remain active even when there are no DR flows that match the rule. Enough of these (NAT based) inspection rules may unnecessarily remain in the switch exhausting its limited memory. Since, in all our experiments, we always observed that the inspection rule (in step 3) was installed in less than 3s, so we set the value of hard time out to be 4s.

For SP redirection rule (step 5 of protocol), controller installs the rule based on client IP, source port and OD IP. Here again, non-DR flows (sharing the same source IP, accessing the same OD as of DR client), shall not match the rule, because different flows have different ports associated with them. Thus, we use an idle timeout with this rule which expires once the flow is inactive for the said duration. Following [62], we set the value of idle timeout to 5s.
2) Proxy–Client traffic reliability, efficiency, and congestion control: The connection between SP and CD is a standard TCP connection, but the "connection" between client and SP merely *appears* to be so. Client–SP packets carry spoofed headers, so they can appear as client–OD traffic to the censor. However, they *cannot* rely on the kernel's TCP stack (due to spoofing). Hence, in order to ensure packets between client and SP are reliably and efficiently delivered (despite the absence of a true connection), *we emulate TCP's congestion and flow control mechanisms at the application layer*. The SP–client traffic, not only bear sequence and acknowledgement numbers corresponding to the initial client–OD traffic, but also mimic TCP sending patterns (varying the sender window sizes, based on acknowledgements and timeouts, as in TCP).
3) Handling clients behind NAT: It must be noted that attempts by multiple clients behind a NAT to use the same OD for DR may lead to service unavailability. This can be explained with an example. Let us assume that for a client C_1, that is behind a NAT, an SP redirection rule (bearing NAT IP_{IP1}, OD IP_{IPO1}, and client's source port) is already installed. At the same time, another client C_2 attempts using the DR service with same OD (IP address: IP_{O1}). This would lead to installation of a new inspection rule for the client C_2, for the same $IP1$ and $IPO1$ pair. Due to this new rule, the DR packets of C_1 would also be redirected to the controller, rather than to the SP. Thus, disrupting the C_1's DR session. As a preventive measure, we do not install any inspection rule for a client IP–OD IP pair, if there

is already a SP redirection rule in place for the same. Thus, the clients behind a NAT device would require trying out different ODs to access DR service.

4) Location of Decoy Station (SP): Previous proposals (except Cirripede and Waterfall) require Decoy Stations to be located close to the Decoy Router. In our design, the Decoy Router is simply an SDN switch, which forwards packets based on flow rules; it can transport DR flows to the SP regardless of its network location. This helps defend against the *Forced Asymmetry* attack discussed ahead in Subsec. 6.2.

6.2 Possible Attacks and Countermeasures

We now consider attack scenarios that may be launched by attackers within our threat model, and describe how SB addresses these threats.

1) *Fingerprinting Attacks*

a) **TLS handshake fingerprinting:** To prevent an attacker from characterizing the TLS handshake signature (*e.g.* the list of ciphers supported), we use the cipher suite of a popular browser Mozilla Firefox[4].

b) **TCP/IP protocol fingerprinting:** An adversary that records TCP/IP header values for traffic to ODs, may use it to distinguish the DR flows from non-DR flows. The DR flows may have different TTL values, window sizes *etc.* (due to packets originating from SP) compared to non-DR flows, making them easily distinguishable. In such a scenario the SP can generate identical TCP/IP characteristics as that of OD, making them relatively indistinguishable from the regular flows.

2) *SB Confirmation Attacks:* The aim of the adversary is to distinguish DR flows from the non-DR flows. To achieve this, an adversary may modify, drop or replay some packets of a TLS connection. However, tampering (or replaying) of regular TLS packets in a normal connection results in the receiver replying with TLS error messages. Since SB hijacks an existing TLS connection, it does not natively respond with such error messages. The adversary might use this distinctive feature to confirm a DR connection. We now describe the kind of attacks such an adversary may launch, along with the preventive measures employed in SB.

a.) Record and Replay (without disrupting regular Internet clients): To confirm usage of DR, censor can record all traffic of a suspected DR user and can try to replay a part of (or complete) connection, at a later time. The first interaction of DR client with SP occurs at step 7. All prior steps corresponding to the original client–OD TLS connection. Thus replaying packets of all the prior steps would not result in confirmation of DR. Hence, the censor may initiate a new DR connection and try to replay *only* the second TLS data packet, carrying the CD URL (step 7 of Fig. 2). Any response from SP, other than an TLS alert, would confirm the use of DR. However, our design is resilient to such an attack as the said packet contains a HMAC derived from the session's HMAC key. Hence, replaying it will lead to failure in HMAC verification. Additionally, none of the subsequent packets can be replayed because of the usage of AES in CBC mode.

b.) Modification and/or Replay (disrupting regular Internet clients): An adversary can attempt to confirm if an arbitrary TLS flow is intended for using DR service. Such an adversary can actively *modify* and/or *drop packets* of an *ongoing* flow. We believe that modifying the packets of such arbitrary TLS flow may prove costly to the adversary. It will disrupt regular Internet communication, eventually degrading the user experience. Such attackers are outside the scope of our threat model (ref. Subsec. 4.1). However, even if the adversary carries out such attacks, we enumerate the possible solutions to mitigate them in Appendix A.3.

3) *Forced Asymmetry:* This attack [51] assumes that traffic from anywhere in the Internet, destined to IP prefixes of censored regime can be (1) classified into traffic originated (or transited), from *Decoy ASes* (ASes which host DR infrastructure) and *non-decoy ASes* and (2) will enter the censorious regime at different locations (*e.g.* BGP routers) in the network using a load balancing technique called as hole punching [25]. *E.g.* if a client accesses OD without DR, its responses will enter at one particular network location. Whereas, if it *pretends* to use OD while using DR, traffic will enter at different location, confirming the use of DR to the censor.

In the existing DR systems (*e.g.* TapDance [60], Slitheen [14] *etc.*) the decoy router and the decoy proxy are placed together within a same decoy AS. The decoy router intercepts the DR requests of the client and forwards it to decoy proxy, which on behalf of client fetches content from the CD and sends it to the client (spoofing as OD). Thus, response packets will always come from decoy AS to the client. This makes the Forced Asymmetry attack successful against such systems.

However, due to SB's modularity, the DR infrastructure is not confined to an AS, because the SP (decoy proxy) can be hosted anywhere on the Internet, even in non-DR ASes. As a consequence, response packets originate from non-DR ASes rather than DR ASes. This

4 We can also use a recent library uTLS [27] to achieve the same.

would result in DR and non-DR traffic entering the censor's network through the same location, making the attack ineffective.

4) *Routing Capable Attacks:* A powerful adversary may disrupt *any* DR system (including SB), by *preventing* traffic originated from its networks to reach the decoy AS. Schuchard *et al.* [51], demonstrated few such attacks, which rely on the censor's ability to control the route taken by packets, either through a *tainted* path (which has DR between client and OD) or a *clean* path (devoid of DR):

a) Crazy Ivan attack: For ongoing client–OD connections (either through a tainted or a clean path), the censor can enforce changes in routing policies such that these connections (DR and non-DR) follow only the clean path. The non-DR flows would not be affected by this change whereas, DR flows would be disrupted.

b) Packet Injection: For a suspected ongoing connection, an adversary can send packets (spoofing as client) along a clean path in order to reveal the actual connection state between client and OD. *E.g.* an adversary could send a crafted TCP packet, with sequence number corresponding to the initial client–OD connection, to the OD. Responses like TCP ACK or Duplicate ACK would indicate absence of DR session, whereas a TCP RST response from the OD would reveal the presence of an ongoing DR connection.

c) Routing around decoys: An adversary might change its routing policies permanently, in order to avoid the DR ASes altogether (by always selecting a clean path), thereby denying DR service.

All the aforementioned attacks rely on the assumption that clean paths can be easily obtained by the adversary. However, recent DR placement strategies (local [35], as well as global [29]) suggest that, obtaining clean paths would be *difficult* for the adversary, considering the massive topological changes (with the associated costs) required to obtain the same [35]. On the contrary, Schuchard in [50] argued that, if the censor manages to obtain a small (yet significant) fraction of clean paths — rerouting its traffic via clean paths would incur heavy economic losses to decoy ASes (which are likely transit ASes that earn revenue by transiting the Internet traffic of their customer ASes). This might build pressure on decoy ASes to remove DRs by inflicting economic losses.

However, to successfully launch former [51] and latter RAD attacks [50], there are some practical challenges. Firstly, a censor needs to find all possible tainted paths from ISPs under its jurisdiction to different Internet destinations, by active probing [51] — a non-trivial exercise for the censor. Secondly, after obtaining tainted and clean paths, censor would have to introduce nationwide BGP policy changes, including changes to routing business relationships, and may lead to network downtimes *etc.*

5) *Delayed SYN:* The client is expected to initiate a TCP connection before the redirection rule expires on the SDN switch (step 3 of the protocol). If client's connection request is delayed and misses this window, intentionally by an adversary (or due to congestion), it would arrive at the OD (without being observed by the controller). The regular TCP and TLS handshakes would ensue. Packets sent by the client would reach OD, not the controller or SP. This does not pose a problem: after the handshakes (TCP and TLS), the subsequent GET request by the client fetches a regular response from the OD. The client immediately realizes that it missed the window, and initiates a new connection request. The censor sees nothing suspicious.

6) *DoS Attacks:* We discuss different ways in which an adversary may abuse the system to deny service to DR and/or non-DR clients. Like others, SB is also vulnerable to such attacks. We also discuss the strategies that are in place, or can be adopted to mitigate them.

a) *Forced Decoy Routing:* We consider the Byzantine attacker that tries to disrupt normal routing for a non-DR user. To do so, the attacker simulates the client side of the SB protocol, while spoofing the source address of the victim (*i.e.* the non-DR user). Eventually, the controller installs redirection rules for the victim's IP address (assuming it to be a DR client). Thus when the victim tries to access OD, its traffic is redirected to the SP which fails to decrypt these packets, and drops them. Thus, by abusing the DR, non-DR traffic is prevented from reaching its intended destination.

However, in our design, the SP redirection rule matches packets based on source IP, destination IP and *source port number*. Thus, to successfully launch the attack, the adversary must *anticipate* the port number the victim will use while connecting to a particular OD; the probability of success of this anticipation is 0.00006 (assuming ephemeral port range = 16383).

b) Memory Exhaustion Attacks: As already mentioned in previous attacks (like Forced Decoy Routing), a powerful adversary can force installation of enough flow rules to exhaust the TCAM memory of SDN switch by pretending to be many legitimate DR clients. To minimize this threat, we incorporate timeouts for different flow rules (step 3 and 5 of protocol). Methods to minimize TCAM memory usage in

SDN [18, 38, 45, 47, 49, 58] can be further used to mitigate this attack.

c) Spamming the Controller: A simple attack would be to flood the controller's email with spam. The adversary could send random emails, or more subtly, emails requesting DR service, from thousands of email addresses. This noise might prevent the controller from detecting legitimate requests. In such scenarios, we resort to issuing a unique email ID for the controller to individual (or groups of) clients. Thus, even if one email ID is spammed, clients can use other email IDs to access DR service. Additionally, similar to Mailet [41], we can also enforce usage limitations on clients or can use Captcha [55] and/or puzzles.

d) Fake Sessions Attack: As a more sophisticated attack, the adversary could send the controller "legitimate-looking" emails, which set up decoy routing sessions with random source IP addresses and ISNs. The controller, receiving an email, would install an inspection rule and begin to analyze the corresponding (irrelevant) flows. *To minimize the impact of this attack, we include a hard timeout with the inspection rule.* A fake email can make the controller inspect unwanted traffic only for the timeout duration, *i.e.* 3s (ref. Subsec. 6.1).

e) Fake DR Request Registration: With an intent to deny some specific users of DR services, an adversary can send an email containing their source IP, some OD-IP and some random ISN. When the actual DR user sends an email, with the same source and OD IP, but a different ISN, the controller stores both the ISNs for matching. Hence, when the DR user sends a TCP SYN to the aforementioned OD IP, it's ISN would match the one indicated in the latter email, and the SP redirection rule would be installed.

An adversary could send multiple such emails with different ISNs. Now the controller would require maintaining a set of ISNs for every client and OD IP pair. Searching through such sets to match ISNs of the incoming SYN packets could marginally increase the overhead at the controller. However, DR services would not be hindered.

Moreover, the presence of the hard timeout of the inspection rule, would force the adversary to keep sending such emails regularly. This could be very expensive for the adversary if the attack has to be carried out for all possible clients. Thus, SB is more resilient to such attacks, unlike other registration based systems (such as Cirrepede and Waterfall of Liberty) where the adversary just needs to register once for a client.

6.3 SDNs and SiegeBreaker

It can be argued that SDNs are primarily deployed in small networks (like data centers), and are not suitable for use on the Internet. However, SDNs have been shown to scale to an entire AS [17, 28, 40], and practical ISP-scale SDN has been demonstrated by Rexford *et al.* [30]. Moreover, there is a recent research that suggests how ISPs can (and should) migrate to SDNs [48]. Further, SB can work even for a non-SDN AS. Friendly non-SDN ISPs could position openflow switches such that they intercept traffic to critical network entities (*e.g.* routers which transport a large fraction of traffic [29]), *without replacing the existing infrastructure.*

Scalability of Controller: Would the SB *controller* become a bottleneck at ISP scale? Recent work [20, 52, 53, 56, 59] suggests that SDN controllers can scale to very large use cases – for instance, `Cuttlefish` [52] proposes a hierarchy of local and global controllers, which can offload tasks to each other, providing higher control plane throughput and better scalability. We suggest that, if needed, such approaches can be used when SB is deployed at scale.

7 Conclusion

In this paper, we present SiegeBreaker, a practical and efficient Decoy Routing system. Using an SDN architecture, SiegeBeaker divides the tasks of Decoy Routing among three loosely-coupled modules. The SDN controller focuses on detecting packets of interest, using an efficient *privacy-preserving* signaling scheme, by reconfiguring SDN switches on-the-fly. The switch then redirects all packets of the Decoy Routing flow to a secret proxy server. Finally, the proxy server communicates with the covert website on behalf of the client, and transmits back the responses *reliably and efficiently.*

In extensive tests involving commercial SDN switches, our prototype shows promising performance — nearly equal to that of direct TCP connections. Additionally, SiegeBreaker's flows uniformly share the available link bandwidth with other non-DR connections. Along with privacy preserving signaling, such performance results show promise for future implementation and deployment by SDN-based networks.

Acknowledgements

We thank our reviewers and our shepherd Amir Houmansadr, for their valuable inputs, which fortified our paper. Further, this research was in part supported by Persistent Systems Ltd., Pune, India.

References

[1] Deterlab: Cyber-Defense Technology Experimental Research laboratory. https://www.isi.deterlab.net/index.php.

[2] Hp10500 series openflow enabled switches data sheet. http://www.hp.com/hpinfo/newsroom/press_kits/2011/InteropNY2011/HP_10500_Data-Sheet.pdf.

[3] Hp3500yl openflow enabled switch data sheet. http://www.curvesales.com/datasheets/switches/Campus-Access/HP-3500-3500-YL-Switch-Series-Datasheet.pdf.

[4] Imap library for python. https://docs.python.org/2/library/imaplib.html.

[5] List of hp sdn switches. https://techlibrary.hpe.com/ie/en/networking/solutions/technology/sdn/portfolio.aspx#.XjhyRtlS_CI.

[6] Openflow Switch Specification. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[7] Ryu- Component Based Software Defined Networking Framework. https://osrg.github.io/ryu/.

[8] Selenium webdriver and ide. https://www.seleniumhq.org/.

[9] Sigebreaker's source code. https://github.com/Piyush825/SiegeBreaker.

[10] Smtp library for python. https://docs.python.org/2/library/smtplib.html.

[11] Tor: Bridges. https://www.torproject.org/docs/bridges.html.en.

[12] The transport layer security (tls) protocol version 1.2. https://tools.ietf.org/html/rfc5246.

[13] Apostol, K. Internet censorship in the arab spring.

[14] Bocovich, C., and Goldberg, I. Slitheen: Perfectly Imitated Decoy Routing Through Traffic Replacement. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2016), CCS '16, ACM, pp. 1702–1714.

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

[16] Burnett, S., Feamster, N., and Vempala, S. Chipping away at censorship firewalls with user-generated content. In *USENIX Security Symposium* (2010), Washington, DC, pp. 463–468.

[17] Caesar, M., Caldwell, D., Feamster, N., Rexford, J., Shaikh, A., and van der Merwe, J. Design and implementation of a routing control platform. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2* (Berkeley, CA, USA, 2005), NSDI'05, USENIX Association, pp. 15–28.

[18] Chuang, C.-C., Yu, Y.-J., Pang, A.-C., and Chen, G.-Y. Minimization of tcam usage for sdn scalability in wireless data centers. In *Global Communications Conference (GLOBECOM), 2016 IEEE* (2016), IEEE, pp. 1–7.

[19] Dingledine, R., Mathewson, N., and Syverson, P. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium* (August 2004).

[20] Dixit, A., Hao, F., Mukherjee, S., Lakshman, T., and Kompella, R. Towards an elastic distributed sdn controller. *ACM SIGCOMM computer communication review 43*, 4 (2013), 7–12.

[21] Ellard, D., Jones, C., Manfredi, V., Strayer, W. T., Thapa, B., Van Welie, M., and Jackson, A. Rebound: Decoy routing on asymmetric routes via error messages. In *Local Computer Networks (LCN), 2015 IEEE 40th Conference on* (2015), IEEE, pp. 91–99.

[22] Ensafi, R., Winter, P., Mueen, A., and Crandall, J. R. Analyzing the great firewall of china over space and time. *PoPETs 2015* (2015), 61–76.

[23] Feamster, N., Rexford, J., and Zegura, E. The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review 44*, 2 (2014), 87–98.

[24] Fraleigh, C., Moon, S., Lyles, B., Cotton, C., Khan, M., Moll, D., Rockell, R., Seely, T., and Diot, S. C. Packet-level traffic measurements from the sprint ip backbone. *IEEE network 17*, 6 (2003), 6–16.

[25] Freedman, M. J., Vutukuru, M., Feamster, N., and Balakrishnan, H. Geographic locality of ip prefixes. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement* (2005), USENIX Association, pp. 13–13.

[26] Frolov, S., Douglas, F., Scott, W., McDonald, A., VanderSloot, B., Hynes, R., Kruger, A., Kallitsis, M., Robinson, D. G., Schultze, S., et al. An isp-scale deployment of tapdance. In *7th USENIX Workshop on Free and Open Communications on the Internet (FOCI)* (2017), USENIX Association.

[27] Frolov, S., and Wustrow, E. The use of tls in censorship circumvention. In *NDSS* (2019).

[28] Fu, J., Sjödin, P., and Karlsson, G. Intra-domain routing convergence with centralized control. *Computer Networks 53*, 18 (2009), 2985–2996.

[29] Gosain, D., Agarwal, A., Chakravarty, S., and Acharya, H. The devil's in the details: Placing decoy routers in the internet. In *Proceedings of the 33rd Annual Computer Security Applications Conference* (2017), ACM, pp. 577–589.

[30] Gupta, A., MacDavid, R., Birkner, R., Canini, M., Feamster, N., Rexford, J., and Vanbever, L. An industrial-scale software defined internet exchange point. In *NSDI* (2016), vol. 16, pp. 1–14.

[31] Herrmann, D., Wendolsky, R., and Federrath, H. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security* (2009), pp. 31–42.

[32] Holowczak, J., and Houmansadr, A. Cachebrowser: Bypassing chinese censorship without proxies using cached content. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 70–83.

[33] Houmansadr, A., Brubaker, C., and Shmatikov, V. The parrot is dead: Observing unobservable network communications. In *Security and Privacy (SP), 2013 IEEE Symposium on* (2013), IEEE, pp. 65–79.

[34] Houmansadr, A., Nguyen, G. T. K., Caesar, M., and Borisov, N. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *Proceedings of the 18th ACM conference on Computer and Communications Security (CCS 2011)* (October 2011).

[35] Houmansadr, A., Wong, E. L., and Shmatikov, V. No direction home: The true cost of routing around decoys. In *NDSS* (2014).

[36] Houmansadr, A., Zhou, W., Caesar, M., and Borisov, N. Sweet: Serving the web by exploiting email tunnels. *arXiv preprint arXiv:1211.3191* (2012).

[37] Houmansadr, A., Zhou, W., Caesar, M., and Borisov, N. Sweet: Serving the web by exploiting email tunnels. *IEEE/ACM Transactions on Networking (TON) 25*, 3 (2017), 1517–1527.

[38] Kannan, K., and Banerjee, S. Compact tcam: Flow entry compaction in tcam for power aware sdn. In *International conference on distributed computing and networking* (2013), Springer, pp. 439–444.

[39] Karlin, J., Ellard, D., Jackson, A. W., Jones, C. E., Lauer, G., Mankins, D. P., and Strayer, W. T. Decoy routing: Toward unblockable internet communication. In *USENIX workshop on free and open communications on the Internet* (2011).

[40] Kotronis, V., Dimitropoulos, X., and Ager, B. Outsourcing the routing control logic: Better internet routing based on sdn principles. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2012), HotNets-XI, ACM, pp. 55–60.

[41] Li, S., and Hopper, N. Mailet: Instant social networking under censorship. *Proceedings on Privacy Enhancing Technologies 2016*, 2 (2016), 175–192.

[42] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev. 38*, 2 (Mar. 2008), 69–74.

[43] McPherson, R., Houmansadr, A., and Shmatikov, V. Covertcast: Using live streaming to evade internet censorship. *Proceedings on Privacy Enhancing Technologies 2016*, 3 (2016), 212–225.

[44] Mohajeri Moghaddam, H., Li, B., Derakhshani, M., and Goldberg, I. Skypemorph: Protocol obfuscation for tor bridges. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM, pp. 97–108.

[45] Mohan, P. M., Truong-Huu, T., and Gurusamy, M. Tcam-aware local rerouting for fast and efficient failure recovery in software defined networks. In *Global Communications Conference (GLOBECOM), 2015 IEEE* (2015), IEEE, pp. 1–6.

[46] Nasr, M., Zolfaghari, H., and Houmansadr, A. The waterfall of liberty: Decoy routing circumvention that resists routing attacks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), pp. 2037–2052.

[47] Obadia, M., Bouet, M., Rougier, J.-L., and Iannone, L. A greedy approach for minimizing sdn control overhead. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on* (2015), IEEE, pp. 1–5.

[48] Poularakis, K., Iosifidis, G., Smaragdakis, G., and Tassiulas, L. One step at a time: Optimizing sdn upgrades in isp networks. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications* (2017), IEEE, pp. 1–9.

[49] Rifai, M., Huin, N., Caillouet, C., Giroire, F., Moulierac, J., Pacheco, D. L., and Urvoy-Keller, G. Minnie: An sdn world with few compressed forwarding rules. *Computer Networks 121* (2017), 185–207.

[50] Schuchard, M. Adversarial degradation of the availability of routing infrastructures and other internet-scale distributed systems. http://hdl.handle.net/11299/182196.

[51] Schuchard, M., Geddes, J., Thompson, C., and Hopper, N. Routing around decoys. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM, pp. 85–96.

[52] Shah, R., Vutukuru, M., and Kulkarni, P. Cuttlefish: Hierarchical sdn controllers with adaptive offload. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)* (2018), IEEE, pp. 198–208.

[53] Song, P., Liu, Y., Liu, T., and Qian, D. Controller-proxy: Scaling network management for large-scale sdn networks. *Computer Communications 108* (2017), 52–63.

[54] United nations general assembly, human rights council thirty-second session, third item. https://www.article19.org/data/files/Internet_Statement_Adopted.pdf.

[55] von Ahn, L., Blum, M., Hopper, N. J., and Langford, J. Captcha: Using hard ai problems for security. In *Advances in Cryptology — EUROCRYPT 2003* (Berlin, Heidelberg, 2003), E. Biham, Ed., Springer Berlin Heidelberg, pp. 294–311.

[56] Wang, C., and Yan, S. Scaling sdn network with self-adjusting architecture. In *2016 IEEE International Conference on Electronic Information and Communication Technology (ICEICT)* (2016), IEEE, pp. 116–120.

[57] Weinberg, Z., Wang, J., Yegneswaran, V., Briesemeister, L., Cheung, S., Wang, F., and Boneh, D. Stegotorus: a camouflage proxy for the tor anonymity system. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM, pp. 109–120.

[58] Wen, X., Yang, B., Chen, Y., Li, L. E., Bu, K., Zheng, P., Yang, Y., and Hu, C. Ruletris: Minimizing rule update latency for tcam-based sdn switches. In *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on* (2016), IEEE, pp. 179–188.

[59] Woo, S., Sherry, J., Han, S., Moon, S., Ratnasamy, S., and Shenker, S. Elastic scaling of stateful network functions. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2018), pp. 299–312.

[60] Wustrow, E., Swanson, C. M., and Halderman, J. A. Tapdance: End-to-middle anticensorship without flow blocking. In *Proceedings of 23rd USENIX Security Symposium (USENIX Security 14)* (San Diego, CA, August 2014), USENIX Association.

[61] Wustrow, E., Wolchok, S., Goldberg, I., and Halderman, J. A. Telex: Anticensorship in the network infrastructure. In *Proceedings of the 20th USENIX Security Symposium* (August 2011).

[62] Zarek, A., Ganjali, Y., and Lie, D. Openflow timeouts demystified. *Univ. of Toronto, Toronto, Ontario, Canada* (2012).

A Appendix

A.1 Tests Over Emulation Environment

Experimental setup: Our experimental topology on DETER consists of ten Linux machines – one of which acted as a SDN switch (running `OpenvSwitch` v2.5.0), two nodes acted as normal routers (forwarding based on routing table lookups) and six host nodes (running Ubuntu 14.04 LTS) functioning as clients and servers. One node functions as the controller, running the `Ryu` [7] SDN controller (v4.15), a popular open-source SDN controller, fully compliant with `OpenvSwitch`. Fig. 13 schematically represents the topology.

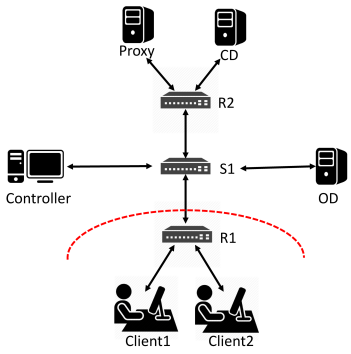

Fig. 13. The topology used for evaluating SiegeBreaker — one SDN switch (`S1`), two routers (`R1`,`R2`), an SDN controller and 6 host nodes all configured on real machines.

Experiments and results: In order to gauge the correctness of our prototype implementation, we tested SiegeBreaker under a variety of working conditions.

We tested SiegeBreaker by downloading relatively small-sized files (< 10MB). This emulates a user's everyday browsing activity. Next, we also tested it against large file sizes, of the order of 1GB, emulating bulk data transfer. Fig. 14 shows the consolidated result of downloading files (varying from 1MB–1GB) using SiegeBreaker and `wget`. Evident from the figure, the download times of both SiegeBreaker and `wget` are comparable. These results indicate that SiegeBreaker works well for both web browsing and bulk downloads.

To compare SiegeBreaker's and `wget`'s behavior in the presence of varying cross traffic, we used one client (Client1 in the topology) to download a file of size 100 MB using SiegeBreaker. Simultaneously, on a shared 100 Mbps link (between R1 and S1), another client (Client2) downloaded a file directly from CD. The second client provided a variable cross-traffic load. This is achieved by

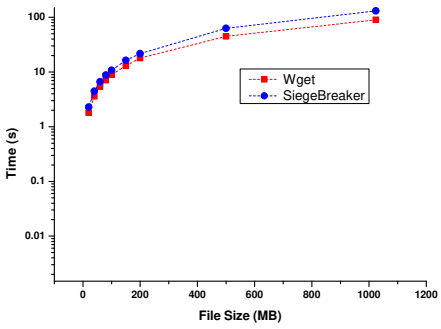

Fig. 14. Comparison of SiegeBreaker and `wget` in terms of download time for file sizes up to 1GB (in log scale).

varying the download request rate of this client (Using the `-rate-limit` option of `wget`) from roughly 8 Mbps to 80 Mbps.

We then repeated the experiment, replacing the SiegeBreaker client with a `wget` client. Fig. 15 shows that the performance of SiegeBreaker is consistently comparable to `wget`, and depicts similar degradation as we increase the background cross-traffic in our tests (which go up to 80 Mbps, *i.e.* 80% of link capacity).

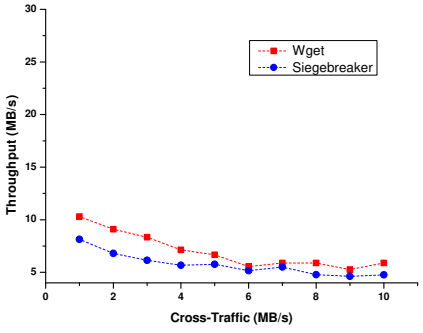

Fig. 15. Comparison of SiegeBreaker and `wget` in presence of varying cross-traffic (on a shared link).

A.2 Incorporating Salient Features of SiegeBreaker in Existing DR Systems

SiegeBreaker is a DR system which utilizes the programmability and reconfigurability of SDNs to redirect and inspect traffic on the fly. This helps in achieving two major goals :

(1) Reducing the overall load on the system by minimizing the amount of traffic to be analysed in order to detect DR flows.

(2) Protecting the privacy of non-DR flows from DR proxy (SP) maintainers.

We now enumerate the feasibility of incorporating these features in existing DR systems. Augmenting ex-

isting signalling schemes with the above goals would require SDN controlled switches acting as DRs. Additionally, they would also require some OOB channel (*e.g.* email) to request DR service. This step ensures that only the *potential* DR flows are analysed. *E.g.* Cirrepede and Waterfall would require to employ SDN controlled switches as DRs, the controller would require to act as the registration server, and would also have to manage the OOB signalling from the client. Thereafter, Cirripede may inspect only the SYN packets of potential DR flows. Moreover, in Waterfall, the controller may need to configure SDN switches to inspect the downstream path instead of the upstream ones.

Telex, Slitheen and Tapdance can also adopt these features. Here, the controller may inspect the respective ClientHello packets, or the incomplete HTTPS requests, to identify DR requests. Once identified, the controller can selectively divert DR flows to their Decoy Stations, which may continue on with its normal functioning by deriving the session key and decrypting the TLS Finished messages *etc.* Slitheen would require the Decoy Station to be co-located with the controller in order to achieve its goal of minimizing the threats of latency based analysis. In Tapdance, after detection, the controller would need to forward all the client-OD traffic to the Decoy Station, and that would need to forward the packets to the OD, to preserve the connection between client and OD, as required in Tapdance.

A.3 SiegeBreaker Confirmation Attacks

In SiegeBreaker, step 5, 6 and 7 can be manipulated to confirm usage of DR. Packets exchanged before step 5 and after step 7 follow standard replay protection mechanism.

Step 5: The adversary can manipulate some bits of the payload sent in this step[5]. In this scenario, the recipient would generate a TLS `bad_record_mac` alert (in accordance with RFC 5246 [12]). However, the controller on receiving this packet has no way to verify if this packet was modified. It will anyways install DR rule and forward this packet to the SP. Unfortunately, the SP also would not be able to verify the message and would never respond with the expected TLS error. This would confirm the adversary of an attempted DR connection. To prevent this attack, we can adopt a scheme as proposed by Tapdance [60]. Here the DR client will covertly leak the client-OD session key to the controller[6]. This will allow the controller to verify if there is any change in the packet and generate appropriate responses.

Step 6: Similar to previous step, some bits of the payload of the TLS data packet (in step 6) can be modified by the adversary. As mentioned, the regular recipient would generate a TLS alert, while a SigeBreaker client would not. Here again the adversary may use this behavior to identify a DR client. To avoid such a situation, the SP includes a HMAC in this packet, generated using the DH key g^{xy}, which can only be derived by the client[7]. The client can thus verify if any modification took place and generate responses accordingly.

Step 7: The adversary can intercept this second TLS data packet (step 7 in Fig. 2), and replace it with its own crafted packet. Reception of legitimate content from CD (via SP), confirms a DR connection. However, since the session keys used to encrypt this packet was derived using the DH parameter (g^x) shared via email. Thus, the adversary cannot derive the same session key, and hence will not be able to craft a legitimate packet.

A.4 SDN Setup

We now describe in detail the setup used in our experiments. The topological diagram of the setups used is shown in Fig. 4 and Fig. 8.

Controlled setup: It involved standard Linux machines as clients (shown as Client 1, 2,...,N). These machines were connected to another Linux machine (with multiple network interfaces) configured to work as a router (shown as R1). This machine was in turn connected to a `HP3500yl` hardware SDN switch. The controller (running Ryu controller application) and OD (two separate Linux machines) were attached directly to the switch. However, the VLANs of the controller and the OD were different as the controller needs to be connected to the switch via a dedicated and isolated channel. SP and CD were also Linux machines connected to `HP3500yl` via another Linux machine acting as a router (shown as R2). The websites were blocked for the client by adding an IPTABLES rule to drop packets destined to CD (on R1).

5 Adversary when tampers the content of the TLS packet, also creates an appropriate TCP checksum, such that the modified packet is still accepted by the receiver's kernel.

6 Client sends an incomplete HTTP GET request in this step, embedding the Client-OD session key.

7 SP has obtained DR client's g^x in step 5.

Internet setup: In this setup, the clients were hosted inside the university campus. The SDN switch was placed outside the purview of the university firewall. The controller and SP machines were directly attached to it. OD and CD were websites hosted on Internet. To capture the performance of large file downloads, we hosted servers on cloud machines, blocked them (via the firewall), and then downloaded the content using SeigeBreaker client.