Your State is Not Mine: A Closer Look at Evading Stateful Internet Censorship

Zhongjie Wang
zwang048@ucr.edu
University of California, Riverside

Yue Cao
ycao009@ucr.edu
University of California, Riverside

Zhiyun Qian
zhiyunq@cs.ucr.edu
University of California, Riverside

Chengyu Song
csong@ucr.edu
University of California, Riverside

Srikanth V. Krishnamurthy
krish@cs.ucr.edu
University of California, Riverside

ABSTRACT

Understanding the behaviors of, and evading state-level Internet-scale censorship systems such as the Great Firewall (GFW) of China, has emerged as a research problem of great interest. One line of evasion is the development of techniques that leverage the possibility that the TCP state maintained on the GFW may not represent the state at end-hosts. In this paper we undertake, arguably, the most extensive measurement study on TCP-level GFW evasion techniques, with several vantage points within and outside China, and with clients subscribed to multiple ISPs. We find that the state-of-the art evasion techniques are no longer very effective on the GFW. Our study further reveals that the primary reason that causes these failures is the evolution of GFW over time. In addition, other factors such as the presence of middleboxes on the route from the client to the server also contribute to previously unexpected behaviors.

Our measurement study leads us to new understandings of the GFW and new evasion techniques. Evaluations of our new evasion strategies show that our new techniques provide much higher success rates of (compared to prior schemes) ≈ 90 % or higher. Our results further validate our new understandings of the GFW's evolved behaviors. We also develop a measurement-driven tool INTANG, that systematically looks for and finds the best strategy that works with a server and network path. Our measurements show that INTANG can yield near perfect evasion rates and is extremely effective in aiding various protocols such as HTTP, DNS over TCP, and Tor in evading the GFW.

CCS CONCEPTS

• **Networks → Network measurement**; **Network privacy and anonymity**; *Middle boxes / network appliances*; • **Social and professional topics → Technology and censorship**; • **Security and privacy →** *Intrusion detection systems*;

IMC '17, November 1–3, 2017, London, UK
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5118-8/17/11…$15.00
https://doi.org/10.1145/3131365.3131374

KEYWORDS

Censorship circumvention, TCP, traffic manipulation, the Great Firewall of China, INTANG

ACM Reference Format:
Zhongjie Wang, Yue Cao, Zhiyun Qian, Chengyu Song, and Srikanth V. Krishnamurthy. 2017. Your State is Not Mine: A Closer Look at Evading Stateful Internet Censorship. In *Proceedings of IMC '17*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3131365.3131374

1 INTRODUCTION

Internet censorship and surveillance are prevalent nowadays. State-level censorship systems such as NSA's PRISM and the Great Firewall (GFW) of China, have the capability of analyzing terabyte-level traffic across the country in realtime. Protocols with plaintext (*e.g.,* HTTP, DNS, IMAP), are directly subject to surveillance and manipulation by the governors [1, 2, 5, 14, 20, 29], while protocols with encryption (*e.g.,* SSH, TLS/SSL, PPTP/MPPE) and Tor, can be identified via traffic fingerprinting, leading to subsequent blocking at the IP-level [13, 31].

The key technology behind these censorship systems is Deep Packet Inspection (DPI) [27], which also powers Network Intrusion Detection Systems (NIDS). As previously reported, most censorship NIDS are deployed "on-path" in the backbone and at border routers [27, 29, 34].

In order to examine application-level payloads, DPI techniques have to correctly implement the underlying protocols like TCP, which is the cornerstone of today's Internet. Ptacek *et al.* [23] have shown that any NIDS is inherently incapable of always reconstructing a TCP stream the same way as its endpoints. The root cause for this is the presence of discrepancies between the implementations of the TCP (and possibly other) protocol at the end-host and at the NIDS. Even if the NIDS perfectly mirrors the implementation of one specific TCP implementation, it may still have problems processing a stream of packets generated by another TCP implementation.

Because of this ambiguity in packet processing, it is possible for a sender to send carefully crafted packets to desynchronize the TCP Control Block (TCB) maintained by the NIDS from the TCB on the receiver side. In some cases, the NIDS can even be tricked to completely deactivate the TCB (*e.g.,* after receiving a spurious RST packet), effectively allowing an adversary to "manipulate" the TCB on the NIDS. Censorship monitors suffer from the same fundamental flaw—a client can evade censorship if the TCB on the censorship monitor can be successfully desynchronized with the one on the server. Different from other censorship evasion technologies such as VPN, Tor, and Telex [32], that rely on additional network infrastructure (*e.g.,* proxy node) [27], TCB-manipulation-based evasion techniques only require crafting/manipulating packets on the client-side and can potentially help all TCP-based application-layer protocols "stay under the radar." Based on this idea, Khattak *et al.* [17] explored several practical evasion techniques against the GFW, by studying its behaviors at the TCP and HTTP layers. The West Chamber Project [25] provides a practical tool that implemented a few of the evasion strategies but has ceased development since 2011; unfortunately none of the strategies were found to be effective during our measurement study. Besides these attempts, there is no recent data point, showing how these evasion techniques work in the wild.

In this work, we extensively evaluate TCP-layer censorship evasion techniques against the GFW. By testing from 11 vantage points inside China spread across 9 cities (and 3 ISPs), we are able to cover a variety of network paths that potentially include different types of GFW devices and middleboxes (see § 3.3 for details). We measure how TCB manipulation can help HTTP, DNS, and Tor evade the GFW.

First, we measure how existing censorship evasion strategies work in practice. Interestingly, we find that most of them no longer work well due to unexpected network conditions, interference from the network middleboxes, or more importantly, new updates to the GFW (different from the model considered previously). These initial measurement results motivate us to construct probing tests to infer the "new" updated GFW model. Finally, based on the new GFW model and lessons learned with regards to other practical challenges in deploying TCP-layer censorship evasion, we develop a set of new evasion strategies. Our measurement results show that the new strategies have a 90% or higher, evasion success rate. We also evaluate how these new strategies can help HTTP, DNS, Tor, and VPN evade the GFW.

In addition, during the course of our measurement study, we design and implement a censorship evasion tool, INTANG, integrating all of the censorship evasion strategies considered in this paper; INTANG is easily extensible to incorporate additional strategies. It requires zero configuration and runs in the background to help normal traffic evade censorship. We plan to open source the tool to support future research in this direction.

We summarize our contributions as the follows:

- We perform the largest measurement study to date, of the GFW's behaviors with TCP-layer censorship evasion techniques.
- We demonstrate that existing strategies are either not working or are limited in practice.
- We develop an updated and more comprehensive model of the GFW based on the measurement results.
- We propose new, measurement-driven strategies that can bypass the new model.
- We measure the success rates of our improved strategies with regards to censorship evasion for HTTP, DNS, VPN, and Tor. The results show very high success rates (> 90 %).
- We develop an open-source tool to automatically measure the GFW's responsiveness, and for censorship circumvention. The tool is extensible as a framework for the integration of additional evasion strategies that may emerge from future research.

2 BACKGROUND

In this section, we provide the background on DPI-based censorship techniques employed by the GFW and discuss previously proposed evasion strategies.

2.1 On-path censorship systems

An "on-path" censorship system wiretaps routers of the ISPs controlled by the censor, makes copies of the packets on the fly and performs analysis in parallel with ongoing traffic. In contrast, an "in-path" censorship system places devices as part of a route, analyzes the traffic and then passes the same to the next hop. The capabilities of an "on-path" system include reading packets and injecting new packets, while an "in-path" system can also discard and/or modify packets. For an "on-path" system, processing time is not critical and thus, it can do more sophisticated analysis; for an "in-path" system, it is critical not to perform heavy analysis that will introduce packet delays. Large-scale censorship systems like the GFW usually deploy the "on-path" design in order to ensure extremely high throughput.

To examine the application-layer content with DPI, a censorship system like the GFW needs to first reassemble TCP streams from the packets. As reported [17], the GFW has a simplified TCP implementation to reconstruct the TCP data flow and pass it to the upper layer for further analysis. The GFW is able to analyze a wide range of application protocols (*e.g.,* HTTP, DNS, IMAP), and can apply its rule-based detection engine to detect sensitive application content.

TCP connection reset is a versatile censorship technique. Due to the "on-path" nature of the GFW, it cannot discard the undesired packets between a pair of end-hosts. Instead it can inject packets to force the connection to shut down, or disrupt connection establishment. Once any sensitive content is detected, the GFW injects RST (type-1) and RST/ACK (type-2) packets to both the corresponding client and the server to disrupt the ongoing connection and sustains the disruption for a certain period (90 seconds as per our measurements). During this period, any SYN packet between the two end-hosts will trigger a *forged* SYN/ACK packet with a wrong sequence number from the GFW, which will obstruct the legitimate handshake; any other packets will trigger forged RST and RST/ACK packets, which will tear down the connection.

According to previous work [3, 25] and our measurements, RST (type-1) and RST/ACK (type-2) are likely from two types of GFW instances that usually exist together. We have encountered some occurences where a type-1 or a type-2 reset occurs individually; thus, we are able to measure their features separately. Type-1 reset has only the RST flag set, and random TTL value and window sizes, while type-2 reset has the RST and ACK flags set, and cyclically increasing TTL value and window sizes.

Once a sensitive keyword detected, the GFW sends one type-1 RST and three type-2 RST/ACK with sequence numbers X, X+1460 and X+4380 (X is the current server-side sequence number). [1] Note

[1]The common size of a full TCP packet is 1460 bytes. Sometimes injected packets can fall behind a server's response and thus, become obsolete and discarded. Sending packets with future sequence numbers can offset this effect to a large extent.

that only type-2 resets entail forged SYN/ACK packets during the 90-second subsequent blocking period; furthermore, only type-2 resets are seen when we split a HTTP request into two TCP packets. From all of the above, we speculate that the type-2 resets are from more advanced GFW instances or devices.

Numerous studies have focused on the TCP connection reset of the GFW. Xu *et al.* [34] perform measurements to determine the locations of the censor devices injecting RST packets. Crandall *et al.* [11] employ latent semantic analysis to automatically generate an up-to-date list of censored keywords. Park *et al.* [20] measure the effectiveness of RST packet injection for keyword filtering on HTTP requests and responses, and provide insights on why filtering based on HTTP responses has been discontinued. Performing TCP connection reset does come with shortcomings. For instance, it is costly to track the TCP state of each and every connection and match keywords against a massive number of TCP packets. It is also not completely resistant to evasion.

DNS poisoning is another common technique used by the GFW [4, 5, 19]. The GFW censors the DNS requests over both UDP and TCP. For a UDP DNS request with a blacklisted domain, it simply injects a fake DNS response; for a TCP DNS request, it turns to the connection reset mechanism. Our measurements also cover *DNS over TCP*.

2.2 Evasion of NIDS and censorship systems

Ptacek *et al.* [23] have systematically studied the vulnerabilities of NIDS in the way that NIDS construct and maintain TCP state. In particular, NIDS maintain a TCP Control Block (TCB) for each live connection to track its state information (*e.g.,* TCP state, sequence number, acknowledgment number, etc.). The goal is to replicate the same exact connection information that exists at both endpoints. However, in practice this is very challenging due to the following factors:

- *Diversity in host information.* Due to ambiguity and updates in TCP specifications, different OS implementations may have very different behaviors in handling TCP packets. For instance, when unexpected TCP flag combinations are encountered, different OSes can behave differently (as how to handle these remains unspecified in the standard). Another example is that RST packet handling has drastically changed over different TCP standards (RFC 793 to RFC 5961).
- *Diversity in network information.* A NIDS usually cannot learn the network topology with respect to the endpoints it is protecting, since the topology itself may change over time. For a LAN, a NIDS can probe and maintain the topology. However, for a censorship system, monitoring the massive scale of the entire Internet is extremely challenging if at all possible. Further, such a system will be unaware of network failures or packet losses. Thus, it cannot judge accurately whether or not a packet has arrived at its destination.
- *Presence of middleboxes.* NIDS usually are not aware of other middleboxes that may be encountered between any pair of communicating endpoints. These middleboxes may drop or even alter packets after the NIDS process them, which makes it even more difficult to reason about how a receiver will behave.

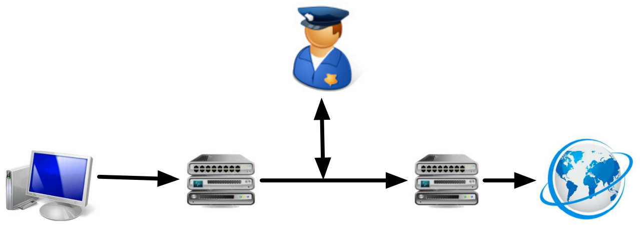

Figure 1: Threat Model

This observation has motivated work on TCP reset attack evasion. For example, Khattak *et al.* [17] manually crafted a fairly comprehensive set of the evasion strategies at the TCP and HTTP layers against the GFW and verified them successfully in a limited setting with a fixed client and server. Unfortunately, there are a large number of factors that were not taken into account (*e.g.,* different types of GFW devices may be encountered on different network paths, various middleboxes may interfere with the evasion strategies by dropping crafted packets).

3 MEASUREMENT OF EXISTING EVASION STRATEGIES

Based on the fundamental limitations of NIDS outlined by Ptacek *et al.* [23], the GFW's modeling by the Khattak *et al.* [17], and the implementation of the West Chamber Project [25], we divide censorship evasion strategies based on TCB-manipulations into three high-level categories, viz., (1) *TCB creation*, (2) *data reassembly*, and (3)*TCB teardown*. In this section, we perform in-depth measurements to evaluate the effectiveness of existing evasion strategies, developed based on the currently known model of the GFW in these categories.

3.1 Threat model

The threat model is depicted in Fig. 1. The client initiates a TCP connection with the server. The GFW establishes a *shadow* connection by creating a TCB and can read from and inject packets to the original connection. Meanwhile, there could be network middleboxes on the path. We refer to the middleboxes between the client and the GFW as *client-side middleboxes* and the middleboxes between the GFW and the server as *server-side middleboxes*.

3.2 Existing evasion strategies

The goal of current evasion strategies (listed below) is to cause the GFW and the server to enter different states (*i.e.,* become desynchronized) by sending specially crafted packets, especially "insertion" packets. These insertion packets are crafted such that they are ignored by the intended server (or never reach the server) but are accepted and processed by the GFW.

TCB Creation. As per previous work [17], the GFW creates a TCB upon seeing a SYN packet. Thus the client can send a SYN insertion packet with a fake/wrong sequence number to create a false TCB on the GFW, and *then* build the real connection. The GFW will ignore the real connection because of its "unexpected" sequence number. The TTL (time to live) or checksum in the insertion packet, is manipulated to prevent the acceptance of the first injected SYN by the server—a packet with a lower TTL value would never reach the intended server and a packet with wrong checksum would be discarded by the server.

Data Reassembly. The data reassembly strategy has two cases: *1. Out-of-order data overlapping.* Different TCP implementations treat overlapping out-of-order data fragments in different ways. Previous work [17] has shown that if the GFW encounters two out-of-order IP fragments with the same offset and length, it prefers (records) the former and discards the latter. However, with regards to out-of-order TCP segments with the same sequence number and length, it prefers the latter (details in [17]). This characteristic with regards to IP fragmentation can be exploited as follows. First, a gap is intentionally left in the payload and a fragment with offset X and length Y, containing random garbage data is sent. Subsequently, the real data with offset X and length Y, containing the sensitive keyword, is sent to evade the GFW (since the GFW is expected to choose the former packet). Finally the gap is filled by sending the real data with offset 0 and length X. To exploit the GFW's handling of TCP segments, we simply switch the order of the garbage data and the real data.

2. In-order data overlapping. When two in-order data packets carrying IP or TCP fragments arrive, both the GFW and the server will accept the first in-order packet that carries a specific fragment (specified by offset/sequence number). One can then craft insertion packets that contain junk data to fill the GFW's receive buffer, while making them to be ignored by the server. For example, one can craft an insertion data packet with a small TTL or a wrong checksum; such packets either never reach or are dropped by the server but are accepted and processed by the GFW.

TCB Teardown. As per the known model, the GFW is expected to tear down the TCB that it maintains when it sees a RST, RST/ACK, or a FIN packet. One can craft such packets to cause the TCB teardown, while manipulating fields such as the TTL or the checksum to ensure that the connection on the server is alive.

3.3 Experimental Setup

We employ 11 vantage points in China, in 9 different cities (Beijing, Shanghai, Guangzhou, Shenzhen, Hangzhou, Tianjin, Qingdao, Zhangjiakou, Shijiazhuang) and spanning 3 ISPs. 9 of these use the cloud service providers (Ailyun and Qcloud) and the other two use home networks (China Unicom). The servers are chosen from Alexa's top websites worldwide. We first filter out the websites that are affected by IP blocking, DNS poisoning, or are located inside China. We exclude the websites that use HTTPS by default, for two reasons. First, HTTPS traffic is not currently censored by the GFW; thus, we can already access them freely without using any anti-censorship technique. Second, if we access these HTTPS websites using HTTP, they send HTTP 301 responses to redirect us to HTTPS, and the sensitive keyword is copied to the *Location* header field of the response. We find that the GFW devices on some paths can in fact detect this in the response packets. This is similar to the HTML response censorship measured in [20]. Furthermore, assuming that GFW devices deployed in a particular autonomous systems (AS) usually are of the same type and version, and configured with the same policy, we choose only one IP from each AS, in order to diversify our experiments by spanning a large set of ASes. By applying filters based on the above rules, and removing a few slow or unresponsive websites, we finally obtain a dataset of 77 websites (from the considered 77 ASes) with Alexa ranks between 41 and 2091. We manually verify that these websites are accessible (outside of China) and are affected by GFW's TCP connection reset upon containing a sensitive keyword, *i.e., ultrasurf*, in the HTTP request. For each strategy and website, we repeat the test 50 times and find the average. Since the GFW will blacklist a pair of hosts for 90 seconds upon the detection of any sensitive keyword, we add intervals between tests when necessary.

3.4 Results

We measure the effectiveness of existing strategies in evading the GFW during April and May in 2017. The results are summarized in Table 1.

Notation: We use the following notation in Table 1: *Success* means that we receive the HTTP response from the server and receive no reset packets from the GFW. *Failure 1* means that we receive no HTTP response from the server nor do we receive any resets from the GFW. *Failure 2* means that we receive reset packets from the GFW, i.e., either RST (type-1) or RST/ACK (type-2).

Results. Our findings are summarized below.

- We find that, possibly because of overloading of the GFW, even if we do not use any evasion strategy, there is a still a 2.8% success rate with regards to retrieving sensitive content. Interestingly this behavior was first documented in 2007 [11] and persists until now.
- We see that TCB creation with SYN does not generally work and has a high "Failure 2" rate (around 89%).
- With regards to data reassembly, we find that (a) out of order data reassembly strategies have both high "Failure 1" and high "Failure 2" rates but (b) sending in-order data to prefill the GFW's buffer results in a much higher success rate (typically > 80%).
- TCB teardown with FIN experiences high "Failure 2" rates while TCB teardown with RST or RST/ACK experience around a 70% success rate, but with a 25% chance trigger reset packets from the GFW.

Evolution of the GFW. We believe that the primary reason for the high failure rates with many existing strategies is because the model of GFW assumed in previous work [17] is no longer valid. While we defer a detailed discussion of how the model has evolved to the next section, we point out here that the "checksum" field is still not validated by the GFW, *i.e.,* a packet with a wrong checksum is still a good insertion packet (the GFW considers it to update its TCB but the server discards it) if there's no interference from network middleboxes. We break down the results with regards to the other reasons why these strategies fail, and analyze them below.

Interference from client-side middleboxes. Client-side middleboxes may drop our insertion packets. Since we manipulate packet fields (*e.g.,* wrong checksum, no TCP flag, *etc.*) to cause the server or server-side middleboxes to discard insertion packets, client-side middleboxes could also discard them. Thus the strategies are voided, and will result in "Failures 2."

Strategy	Discrepancy	w/ sensitive keyword			w/o sensitive keyword	
		Success	Failure 1	Failure 2	Success	Failure 1
No Strategy	N/A	2.8%	0.4%	96.8%	98.9%	1.1%
TCB creation with SYN	TTL	6.9%	4.2%	88.9%	95.3%	4.7%
	Bad shecksum	6.2%	5.1%	88.7%	93.5%	6.5%
Reassembly out-of-order data	IP fragments	1.6%	54.8%	43.6%	45.1%	54.9%
	TCP segments	30.8%	6.5%	62.6%	92.8%	7.2%
Reassembly in-order data	TTL	90.6%	5.7%	3.7%	95.1%	4.9%
	Bad ACK number	83.1%	7.5%	9.5%	93.5%	6.5%
	Bad checksum	87.2%	1.9%	10.8%	98.4%	1.6%
	No TCP flag	48.3%	3.3%	48.4%	97.1%	2.9%
TCB teardown with RST	TTL	73.2%	3.2%	23.6%	94.7%	5.3%
	Bad checksum	63.1%	7.6%	29.3%	89.5%	10.5%
TCB teardown with RST/ACK	TTL	73.1%	3.2%	23.7%	97.1%	2.9%
	Bad checksum	68.9%	1.9%	29.2%	98.2%	1.8%
TCB teardown with FIN	TTL	11.1%	1.0%	87.9%	99.4%	0.6%
	Bad checksum	8.4%	0.8%	90.7%	99.0%	1.0%

Table 1: Probing the GFW from 11 vantage points with 77 websites; experiments are repeated 50 times for each client/server pair.

Packet Type	Aliyun(6/11)	QCloud(3/11)	China Unicom SJZ(1/11)	China Unicom TJ(1/11)
IP fragments	Discarded	Reassembled	Reassembled	Reassembled
Wrong TCP checksum	Pass	Pass	Pass	Dropped
No TCP flag	Pass	Pass	Pass	Dropped
RST packets	Pass	Sometimes dropped	Pass	Pass
FIN packets	Sometimes dropped	Pass	Dropped	Dropped

Table 2: Client-side middlebox behaviors

On the other hand, some NAT or state/sequence checking firewalls deployed on the client-side of the network might intercept and accept the insertion packets and change their maintained connection state. In such cases, later packets will not go through these middleboxes, resulting in "Failures 1." For example, if a RST packet tears down the connection on a client-side middlebox which it traverses, the middlebox blocks later packets on that connection.

Some client-side middleboxes may discard IP fragments (wrt data reassembly strategies) and cause "Failures 1." Others buffer and reassemble them into a whole IP packet and this might cause "Failures 2" depending on the implementation of the middlebox.

We probed for client-side middleboxes from all our 11 vantage points trying to connect with our own servers. As shown in Table 2, we found that our 6 clients using Aliyun were unable to send out IP fragments. One can conclude within reason that Aliyun has configured its middleboxes to discard certain kinds of IP fragments. We found that connections from the other 5 nodes encounter client-side middleboxes, which reassemble the IP fragments into a full IP packet containing the original HTTP request; thus these packets were deterministically captured by the GFW. Since we found that most of the routers and/or middleboxes interfere with IP-layer manipulations, we argue that this is not as *generally* applicable as TCP-layer manipulations for evasion.

The vantage point in Tianjin China Unicom has client-side middleboxes that drop packets with wrong TCP checksums or containing no TCP flag; thus these two strategies did not work at that point. Finally, we found Aliyun sometimes drops FIN insertion packets and QCloud sometimes drops RST insertion packets. Both the clients in Shijiazhuang and Tianjin (China Unicom) have client-side middleboxes that drop FIN insertion packets.

Interference from server-side middleboxes. Server-side middleboxes only affect the server but not the GFW. Our insertion packet may terminate the connection or change the connection state on the server-side middleboxes causing later packets to be blocked by the middleboxes. This will cause "Failures 1." To verify interference from server-side middleboxes, we need to either control the server or set up our own server on the same path behind those middleboxes, which are infeasible in practice for all our targets, *i.e.,* the Alexa's top websites.

Other reasons for failures. There could be a few other reasons for observing failures of the two types. Network or server failures although rare could occur. We performed microscopic studies of our failure cases and list the cases that we observed below.

Variations in server implementations. We find that with some server implementations (*e.g.,* Linux versions prior to 3.8), a data packet under "in-order data overlapping strategy" carrying no TCP flag can sometimes be accepted by the server and thus causes "Failures 1." With the "out-of-order data overlapping strategy," a server might accept the junk data (just like the GFW) and discard the correct packet.

Network dynamics. Since routes are dynamic and could change unexpectedly, the TTL values used in the insertion packets to prevent them reaching the server could be incorrect. As a result, they may reach the server and disrupt the connection (Failures 1). In other cases, the insertion packets might not reach the GFW and lead to "Failures 2." We also found that packet losses on the network could affect the insertion packets and cause "Failures 2." We cope with such dynamics by repeating the sending of the insertion packets thrice with 20ms intervals.

Summary. Our measurement uses real web servers instead of controlled servers in order to represent cases of daily web browsing. The results demonstrate the complexity induced by many factors (*e.g.,* middlebox interference, server diversity, network diversity, *etc.*). We showcase the overall success rates with existing evasion strategies and enumerate possible reasons for the failure cases. To fully untangle the factors causing failures and to quantify the impact of each, more in-depth analysis and controlled experiements are required (*e.g.,* using controlled replay server as in [18]), which we leave for future work.

4 EVOLVED GFW BEHAVIORS

As alluded to in § 3, high failure rates were experienced even if we eliminated the effects from middleboxes, server implementations, and network dynamics. To understand the root cause, we take a closer look and argue that this is due to evolved GFW behaviors that break many prior assumptions. Based on our measurements, we hypothesize these new behaviors as follows. To verify these hypotheses, in § 7 we design and extensively evaluate new evasion strategies.

Prior Assumption 1: *The GFW creates a TCB* only *upon seeing a SYN packet.*

To test this assumption, we used pairs of clients and servers under our control, and executed partial TCP 3-way handshakes (*e.g.,* intentionally omitting the SYN, SYN/ACK and/or ACK) followed by a HTTP request with a sensitive keyword. If a correct TCB was created on the GFW, the HTTP request would trigger TCP reset packets from it. First, our results confirmed that the GFW still creates a TCB upon seeing a SYN packet as described in [17]. Second and more interestingly, we found that the GFW also creates a TCB upon seeing a SYN/ACK packet *without* the SYN packet. We speculate that the GFW has evolved to incorporate this feature to counter SYN packet losses. Given these, we hypothesize that the GFW exhibits the following new behavior.

Hypothesized New Behavior 1: *The GFW creates a TCB not only upon receiving SYN packets, but also SYN/ACK packets.*

Prior Assumption 2: *The GFW uses the sequence number in the first SYN packet to create a TCB, and ignores later SYN packets during the lifetime of the TCB.*

This assumption is based on the rationale that the GFW mimics a normal TCP implementation. Our closer look revealed that it does not. From the results in § 3, we see that the TCB creation with a SYN insertion packet failed in most cases. This leads us to re-examine this case. We send multiple SYN packets among which only one has the "true" sequence number, and then send a sensitive HTTP request. However, no matter where we put the "true" SYN packet, the GFW can always detect the later sensitive keyword. We hypothesize that this could be because of any of three possible reasons:

- (1) the GFW establishes multiple TCBs, one for each SYN packet;
- (2) the GFW enters a "stateless mode", in which it checks every individual packet instead of re-assembling the data first (and check for a sensitive keyword);
- (3) the GFW uses the sequence number in the HTTP request to re-synchronize its TCB.

To check (1), we set the sequence number in the HTTP request to be a "out-of-window" value with respect to the sequence numbers in the SYN packets; however, we find that the GFW can still detect the keyword. To examine (2), we split the sensitive keyword into halves, each of which by itself is *not* a sensitive keyword; however, we find that the GFW can still detect it. For (3), before sending the HTTP request, we send some random data with a "false" sequence number, and then we send the HTTP request with "true" sequence number; the GFW cannot detect it in this case. This suggests that the GFW re-synchonrizes its TCB with the sequence number in the random data, and thus, ignores the later HTTP request because of its out-of-window sequence number. This validates hypothesis (3) that the GFW enters a "re-synchronization state" upon seeing multiple SYN packets. We further validate this extensively in § 7.

Besides multiple SYN packets, we found that multiple SYN/ACK packets or a SYN/ACK packet with an incorrect acknowledgment number can also cause the GFW to enter the re-synchronization state.

Next, we try to find out "which packet the GFW uses to re-synchronize its TCB once in *re-synchronization state*." From the previous experiement, we learn that the GFW re-synchronizes using data packets from the client to the server. Thus, instead, we try to use data packets from the server to the client; in addition, we try pure ACK packets without data in both directions. We find none of these packets affect the GFW. However, we do find that a SYN/ACK packet from the server to the client can cause re-synchronization. We admit that the cases we found may not be complete but it is hard to enumerate an exhaustive set of these cases. However, our measurements lead us to a better understanding of the GFW behavior than what exists today and leads us to the following new hypothesis.

Hypothesized New Behavior 2: *The GFW enters what we call the "re-synchronization state", where it re-synchronizes its TCB using the information in the next SYN/ACK packet from server to client or data packet from client to server upon experiencing any of the following three cases: (a) it sees multiple SYN packets from client-side, (b) it sees multiple SYN/ACK packets from server-side, or (c) it sees a SYN/ACK packet with an acknowledgment number different from the sequence number in the SYN packet.*

Prior Assumption 3: *The GFW tears down a TCB when it sees a RST, RST/ACK or FIN packet.*

The results in § 3 suggest that the evolved GFW generally does not tear down a TCB merely upon seeing FIN packets. At the same time, we also observed high failure rates of above 20% with our RST and RST/ACK insertion packets. A closer look suggests that this probably is due to "Hypothesized New Behavior 2." More specifically, we found that when the GFW is in the newly discovered "re-synchronization state", its TCB sometimes cannot be torn down with RST or RST/ACK packets. To verify this, we force the GFW to enter the re-synchronization state using one of the techniques above, and then immediately send a RST packet and a HTTP request with sensitive keyword. However, the GFW sometimes can still detect it. We repeated the experiement at different times with multiple pairs of clients and servers, and found inconsistency between different measurements across pairs at different times. The overall success rate is roughly 80%, and for a specific client-server pair, the GFW's behavior is usually consistent during a certain period (although not always across periods). We are unable to unearth the explicit reason behind at this time; we conjecture that it is due to dynamics with regards to the heterogeneity in the types of GFW encountered and the complexity of interactions among different GFW instances and middleboxes. We discuss this further in § 8.

In addition, we performed extensive measurements wherein we sent a RST packet between the SYN/ACK and the ACK packet of the 3-way handshake, and also after the 3-way handshake. We found that in both cases the TCB sometimes is not torn down but the RST packet caused the GFW to enter the re-synchronization state; further, we find that this happens way more frequently for the former case (the exact reason for the discrepancy remains unknown). These observations lead to the following new hypothesis.

Hypothesized New Behavior 3: *Upon receiving a RST or RST/ACK packet, the GFW may enter the re-synchronization state instead of tearing down the TCB.*

5 NEW WAYS TO EVADE THE GFW

In this section, we discuss new opportunities for evasion from two perspectives. First, based on the new hypothesized behaviors of the GFW, we propose new evasion strategies. Second, we attempt to systematically discover new insertion packets (besides wrong checksum or small TTL).

5.1 Desynchronize the GFW

First of all, we describe a building block to counter the re-synchronization state in the GFW. It is helpful in supporting our new evasion strategies, which are discussed next. Specifically, when we expect that the GFW is in the re-synchronization state (this can be forced), we send a insertion data packet with a sequence number that is out of window. Once the GFW synchronizes with the sequence number in this insertion packet, subsequent legitimate packets of the connection will be perceived to have sequence numbers that are out of window, and thus be ignored by the GFW. We say that now the GFW is *desynchronized* from the connection. Note that the insertion data packet is ignored by the server since it contains an out-of-window sequence number.

Desynchronizing the GFW drastically helps improve the "TCB Teardown" and the "In-order Data Overlapping" strategy that still work relatively well but occasionally experience undesired high "Failure 1" and "Failure 2" rates.

5.2 New Evasion Strategies

Our evasion strategies are primarily based on exploiting the newly discovered state of the GFW. We propose two new evasion strategies along with improvements to two existing strategies.[2] We evaluate these extensively in § 7. The two new strategies are as follows:

Resync + Desync. To coerce the GFW into entering the re-synchronization state, the client sends a SYN insertion packet after the 3-way handshake. Subsequently, the client sends a 1-byte data packet containing an out-of-window sequence number to desynchronize the GFW. This is then followed by the real request. Note that the SYN insertion packet cannot be sent prior to receiving the SYN/ACK packet, as the GFW will eventually resynchronize the expected client-side sequence number based on the ACK number of the SYN/ACK. In addition, the SYN insertion packet should take a sequence number outside of the expected receive window of the server (as in older Linux this can cause the connection to reset). Newer versions of Linux will never accept such a SYN packet regardless of its sequence number and will simply respond with a challenge ACK [7]. In addition, we can craft the insertion SYN packets with small TTL in case the server or middleboxes interfere.

TCB Reversal. As discussed, the GFW currently only censors traffic from the client to the server (*e.g.,* HTTP/DNS requests), and the censorship of HTTP response has been discontinued except in a few rare cases [20]. When the GFW first sees a SYN/ACK, it assumes that the source is the server and the destination is the client. It creates a TCB to reflect that this is the case. It will now monitor data packets from the server to the client (mistakenly thinking that it is monitoring data packets from the client to the server). To exploit this property, the client will first send a SYN/ACK insertion packet. It later performs the TCP three way handshake in a normal way. The GFW will ignore these handshake packets since there already exists a TCB for this connection. Note that the SYN/ACK insertion packet has to be crafted with care. In normal cases, the server responds with a RST which causes a teardown of the original TCB at GFW. To address this, one of the discrepancies (*e.g.,* lower TTL) will need to be used in the insertion packet. In addition, we point out that here the SYN/ACK and subsequent SYN packet from the client do not trigger the GFW to enter the resynchronization state.

5.3 New Insertion Packets

All GFW evasion strategies require injecting additional packets or modifying existing packets to disrupt the TCP state maintained on GFW [17, 23]. Insertion packets are especially handy as they are the most suitable for supporting evasion strategies against the GFW.

As alluded to in §3, insertion packets can be tricky to craft. They may fail because of many reasons such as network dynamics, routing asymmetry, obscure network middleboxes, and variations in server TCP stacks. Our observation is that none of the insertion packets are universally good. This motivates us to discover additional insertion packets that may be viable and complementary to existing insertion packets.

[2]For brevity we only describe the new strategies in this section and leave the detailed discussion of improved strategies to §7.

TCP State	GFW State	TCP Flags	Condition
Any	Any	Any	IP total length > actual length
Any	Any	Any	TCP Header Length < 20
Any	Any	Any	TCP checksum incorrect
SYN_RECV	ESTABLISHED/RESYNC	RST+ACK	Wrong acknowledgement number
SYN_RECV/ESTABLISHED	ESTABLISHED/RESYNC	ACK	Wrong acknowledgement number
SYN_RECV/ESTABLISHED	ESTABLISHED/RESYNC	Any	Has unsolicited MD5 Optional Header
SYN_RECV/ESTABLISHED	ESTABLISHED/RESYNC	No flag	TCP packet with no flag
SYN_RECV/ESTABLISHED	ESTABLISHED/RESYNC	FIN	TCP packet with only FIN flag
SYN_RECV/ESTABLISHED	ESTABLISHED/RESYNC	ACK	Timestamps too old

Table 3: Discrepancies between GFW and server on ignoring packets – candidate insertion packets

The ideal solution to discovering insertion packets is to obtain a precise TCP model for the GFW, the server, and network middleboxes that can be fed into an automated reasoning engine (to see what kinds of packets can qualify as insertion packets). However, since the GFW is a blackbox with only one observable feedback attribute (viz., the RST injection), it is quite hard to infer its internal state accurately and completely. The evolved GFW model that we infer in §4 is also unlikely to be complete. Therefore, even if one were to leave network middleboxes aside, the problem is very challenging.

Our solution is as follows: instead of attempting to model the GFW accurately, we first model the servers (*e.g.,* popular Linux and FreeBSD TCP stacks) using "ignore" paths analysis. By this we mean that we want to identify and reason about points in a server's TCP implementation which cause it to ignore received packets. Specifically, for an incoming packet, we analyze all possible program paths that lead to the packet being either discarded completely, or "ignored" possibly with an ACK sent in response. An example of the first case is a packet with an incorrect checksum; the second case can be a data packet with an out-of-window sequence number, which triggers a duplicate ACK [21]. In both cases, the TCP state (*e.g.,* the next expected sequence number) of the host (server) remains unchanged. After we derive this server model, we use it to develop probing tests against the GFW.

For open source operating systems such as Linux, this can be achieved through static analysis similar to what is done in PacketGuardian [8]. The challenge is to manually identify all program points where "ignore" events occur. Once the ignore paths are identified, the constraints that lead to each path need to be computed, and used to guide test packets against the GFW. Once we identify cases where the packets are "accepted" by the GFW, *i.e.,* the GFW updates its TCB according to the information in the packet, we can conclude that such packets are effective insertion packets (note that we have not yet considered interference from network middleboxes).

During the analysis, we only need to consider the TCP states that still have the potential to receive data, *i.e.,* TCP_LISTEN, TCP_SYN_RECV, TCP_ESTABLISHED. For instance, we omit the TIME_WAIT state because the server can no longer receive data in this state and it is fruitless to understand its ignore paths. After we generate the ignore paths of the server for each TCP state, we first generate a sequence of packets that lead to the specific TCP state; then for the set of constraints generated for each ignore path, we generate one or more test packets (as candidate insertion packets). Note that each ignore path will lead to a unique reason for why the packet will be ignored by the server (*e.g.,* either wrong checksum or invalid ACK, but never both). Ptacek *et al.* [23] used a similar approach to study the FreeBSD TCP stack, which is unfortunately too old to be applicable. In contrast, we study the latest Linux TCP stack, which has many new behaviors. Further, we improve the methodology by pruning a number of "ignore" paths in irrelevant TCP states such as TIME_WAIT, as well as correlating the "ignore" cases with middlebox behaviors.

As a demonstration, we conduct such an analysis of Linux kernel version 4.4. In Table 3, we list the confirmed cases in which Linux ignores a packet but the GFW does not. We also try to compare the server state with the GFW state to make the discrepancies more clear. Note that this is a more complete list than what was previously reported [17, 23], demonstrating the advantage of our systematic analysis. For instance, the finding includes two new insertion packets:

1) RST/ACK packets with incorrect ACK number are ignored by the server in TCP_RECV state but GFW will accept such a packet and change its state to either TCP_LISTEN (previous state terminated) or TCP_RESYNC, depending on the GFW model.

2) Packets with unsolicited MD5 headers are ignored by the server (if no prior negotiation of optional MD5 authentication has been done) while GFW will process the packet as normal.

The MD5 header [15] discrepancy can be exploited in an insertion packet with any TCP flag. For example, this can be used in a RST packet to tear down the TCB on the GFW, or in a data packet to fool the GFW into changing its maintained client sequence number.

Note that we intentionally omit the analysis of data overlapping (for processing out-of-order and overlapping data packets) discrepancies as it has been understood that different OSes may employ different strategies [23] and thus it may not lead to a safe insertion packet.

Cross-validation with network middleboxes. Even though the insertion packets generated from the analysis work well according to our experiments, they may not play well with middleboxes. Note that IP layer discrepancies such as wrong IP checksum, IP optional header, and IP header length can be used under all TCP states for all TCP flags, but packets with such properties are often dropped by routers or middleboxes. The only feature that we find useful is the one where the "IP total length" is larger than the "actual packet length" (listed in Table 3); however, packets with this feature may still be checked and dropped by some middleboxes. Even insertion packets that leverage TCP layer discrepancies (such as those relating to improper TCP header lengths or the wrong TCP checksum) may still be dropped by middleboxes, especially in cases where the perturbation applies to all TCP states and flags. The only exceptions are insertion packets leveraging the unsolicited MD5 header; these are never dropped by the middleboxes we encounter during our experiments (presumably because it requires a stateful firewall middlebox to understand when such packets should be dropped).

The remainder of the insertion packets can be useful only for data packets. Effective control packets cannot be crafted with these; for instance, when the server is in the ESTABLISHED state, even if the RST/ACK has a wrong ACK number or old timestamp, it will still be able to reset the connection successfully. According to our experiments, we have not encountered middleboxes that drop packets with unexpected MD5 options, old timestamps, or incorrect ACK numbers.

Cross-validation with other TCP stacks. It is difficult, if not impossible, to exhaustively test the ignore paths of all deployed TCP stacks. We cross-validate the ignore paths of Linux kernel 4.4 with several other popular Linux versions, including 4.0, 3.14, 2.6.34, and 2.4.37. We summarize the results here:

- In Linux 3.14, when a connection is in the ESTABLISHED state, an incoming packet with a SYN flag will be ignored, while the new GFW model will accept it.
- In Linux 2.6.34 and 2.4.37, when a connection is in ESTABLISHED state, an incoming packet without a set ACK flag will not be ignored. Instead, a data packet without the ACK flag will in fact be accepted. This indicates that such an insertion packet will not work against older Linux versions.
- In Linux 2.4.37, an incoming packet with an unsolicited MD5 header will not be ignored. This is due to the fact that older Linux versions have not implemented the feature proposed in RFC 2385 [15]. Upon closer inspection, the MD5 option check on the server can be turned off via kernel compilation options and therefore the corresponding insertion packet in fact may not always work.

This shows most insertion packets are applicable to a wide range of Linux operating systems, with some minor exceptions (if the encountered Linux version is too old). As Linux is dominant in the server market [26], we envision that evasion strategies built on top of these insertion packets will work well. Indeed, as we show in §7, our GFW evasion success rate is extremely high if we are to leverage these insertion packets properly. To discover additional discrepancies and perform automatic "ignore path" analysis, we plan to use selective symbolic execution in the future (*e.g.,* S2E [9]). We leave a more rigorous analysis of TCP stacks of other Linux versions and operating systems, including closed-source OS like Windows Server, to our future work.

6 INTANG

All the strategies described in § 3 and § 4, are together integrated in a unified *measurement driven censorship evasion tool* we call INTANG. [3] The implementation contains roughly 3.3K lines of C code and some analysis scripts written in Python. INTANG is designed as an extensible framework that supports add-on strategies. The components of INTANG are depicted in Fig. 2.

[3]INTANG source code is publicly available at https://github.com/seclab-ucr/INTANG.

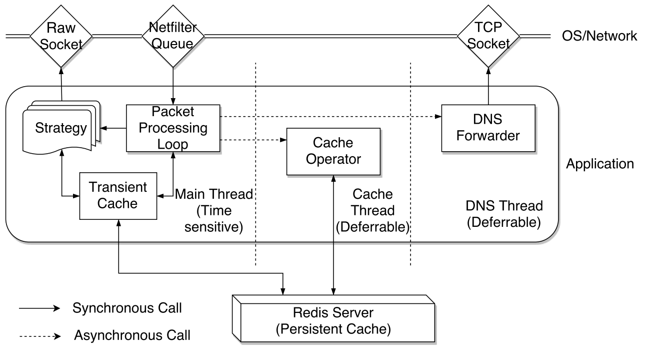

Figure 2: INTANG and its components

Overview. INTANG's functionalities are divided into three threads, viz., the main thread, the caching thread, and the DNS thread. The main thread is time-sensitive, and all time-consuming operations are pushed to the other two threads. The main thread runs a packet processing loop which intercepts certain packets using the netfilter queue [6] and injects insertion packets using raw sockets. While the packets are being processed, they are held in the queue *i.e.,* are not sent out until the processing is complete.

When a new connection is initiated, INTANG chooses the most promising strategy based on historical measurement results (with the help of caching), to a particular server IP address. Upon the completion of a successful trial, it caches the strategy ID along with the four-tuple of the connection in memory. When it later receives further packets associated with the four-tuple, it will invoke the callback functions of the strategy to process the incoming and outgoing packets. Usually, only a small set of specific packets (e.g. SYN/ACK packet, HTTP request) are relevant to each strategy and need monitoring (as discussed earlier).

DNS forwarder. The DNS thread is a specialized thread that aims at converting DNS requests over UDP to DNS requests over TCP. As mentioned in § 2.1, TCP-layer evasion not only helps with evading censorship on HTTP connections, but can also support the evasion of DNS poisoning by GFW. For this purpose, a simple DNS forwarder is integrated within INTANG. It converts each DNS over UDP request to a DNS TCP request and sends it to an unpolluted, public DNS resolver (likely outside of China). We apply the same set of strategies for the TCP connection that carries DNS requests and responses, to prevent the GFW from resetting the connection upon detecting a censored domain in the request. The main thread intercepts outgoing DNS UDP requests, which may contain sensitive domain names and redirects such requests to the DNS thread that does the forwarding. When a DNS TCP response is received, it will be converted back to a DNS UDP response and processed normally by the application. So it is completely transparent to applications. We have probed GFW with Alexa's top 1 million domain names to generate a list of poisoned domain names using the same method as in [12].

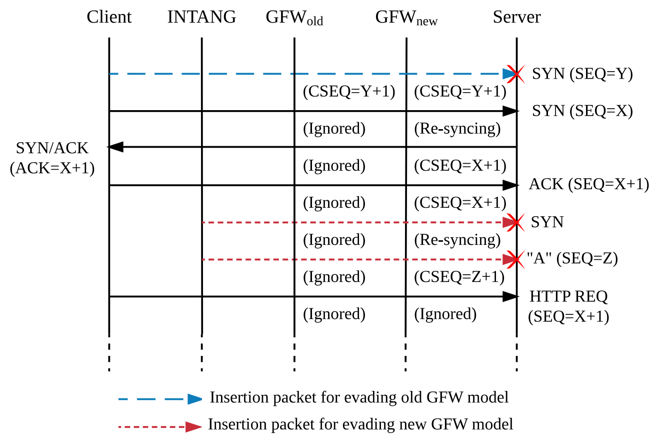

Figure 3: Combined Strategy: TCB Creation + Resync/Desync

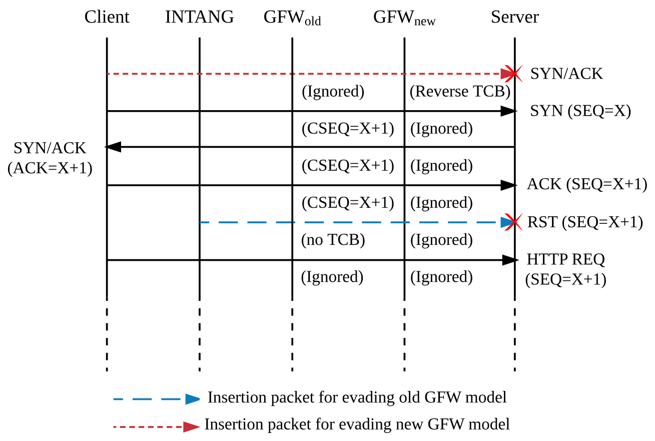

Figure 4: Combined Strategy: TCB Teardown + TCB Reversal

Strategies. Each evasion strategy dictates specific interception points (*i.e.,* the types of packets to intercept) and the corresponding actions to take at each point (*e.g.,* inject an insertion packet). A new strategy can be derived from our suite of basic strategies by implementing new logic in the callback functions registered as interception points. A strategy can decide on whether to accept or to drop an intercepted packet, and can also modify the packet. It can craft and inject new packets as well.

Caches. INTANG employs Redis [24] as an in-memory key-value store. Redis provides desirable features like data persistency, event-driven programming, key expiration, etc. We also maintain in the main thread, a transient Least Recently Used (LRU) cache implemented using linked lists and hash tables (to reduce Redis store access latency that typically involves inter-thread or inter-process communications). Caching allows us to understand the effectiveness of the strategies against different websites and converge on the best one quickly. Of course, to counter changes in the network or the server, the cached record is retained only for a certain period of time before expiration. We omit the details of cache management in the interest of space.

7 EVALUATION

We now extensively evaluate the hypothesized new behaviors of the GFW discussed in §4 using the new strategies described in §5 and our tool INTANG. We use the same 11 vantage points and 77 web servers as discussed in §3; unless otherwise specified, all other measurement settings remain the same to ensure the consistency of the results. The experiments were conducted during April and May, 2017. In addition, since the GFW not only censors outbound traffic but also inbound traffic (both are client-to-server traffic),[4] we conduct measurements from 4 vantage points outside China, viz, in US, UK, Germany, and Japan, using instances on Amazon EC2, to targets inside China. This dataset includes top 33 Chinese websites chosen from the same Alexa's top 10,000 websites using the same method as in § 3.3 except they are inside China. By doing the bi-directional evaluation, we are in hope to examine if our new hypotheses/strategies work well for both directions and the implementations/policies of the GFW in both directions are the same.

[4]A possible reason for doing this could be achieving bi-directional information barriers such as censoring what outsiders can see or restricting certain services, *e.g.,* VPN.

7.1 Evading HTTP censorship

There are 4 basic strategies that we evaluate in this subsection. These include two improved strategies based on previous strategies. These still worked but had high "Failure 1" and "Failure 2" rates. Specifically, they are *TCB Teardown with RST* and *In-order Data Overlapping*. The other two are new strategies viz., *Resync-Desync* and *TCB Reversal*. Note that these latter strategies explicitly leverage the new features that only exist in the evolved GFW model. We combine them with the aforementioned existing strategies that work for the old GFW model, in order to defeat both GFW models (*i.e.,* the objective is to defeat the GFW regardless of whether an old GFW model or an evolved model is encountered, or both).

Making old strategies robust. We make the *TCB Teardown with RST* strategy more robust by integrating within it, the sending of a "desynchronization packet" mentioned in §4. We send this desynchronization packet right after the RST packet(s) and before the legitimate HTTP request, to address the case wherein the GFW enters the "resynchronization state" due to the RST packets. We improve the reliability of the *In-order Data Overlapping* strategy by using more carefully chosen insertion packets to reduce potential interference from middleboxes, or because of hitting the server.

Accounting for both old and new GFW models. We combine the *Resync-Desync* strategy with the *TCB Creation with SYN* strategy. The latter can evade the old GFW model by causing the creation of a false TCB, while the former can desynchronize the evolved GFW model by forcing them into the resynchronization state first. Specifically, as illustrated in Fig. 3, we will send two SYN insertion packets (both with wrong sequence numbers), one before the legitimate 3-way handshake and one after, and followed by a desynchronization packet and then the HTTP request. Note that the first SYN insertion packet followed by the legitimate SYN does also cause an evolved GFW to enter the resynchronization state; however, it is later resynchronized with SYN/ACK packet. We therefore need another SYN insertion packet after the handshake to cause the

Vantage Points	Strategy	Success			Failure 1			Failure 2		
		Min	Max	Avg.	Min	Max	Avg.	Min	Max	Avg.
Inside China	Improved TCB Teardown	89.2%	98.2%	**95.8%**	1.7%	6.7%	**3.1%**	0.0%	5.4%	**1.1%**
	Improved In-order Data Overlapping	86.7%	97.1%	**94.5%**	2.9%	8.9%	**4.4%**	0.0%	5.2%	**1.1%**
	TCB Creation + Resync/Desync	88.5%	98.1%	**95.6%**	1.9%	7.0%	**3.3%**	0.0%	5.5%	**1.1%**
	TCB Teardown + TCB Reversal	90.2%	98.2%	**96.2%**	1.7%	5.6%	**2.6%**	0.0%	5.7%	**1.1%**
	INTANG Performance	93.7%	100.0%	**98.3%**	0.0%	3.0%	**0.9%**	0.0%	3.5%	**0.6%**
Outside China	Improved TCB Teardown	85.6%	92.9%	**89.8%**	4.6%	7.6%	**6.8%**	0.3%	6.8%	**3.5%**
	Improved In-order Data Overlapping	89.4%	96.0%	**92.7%**	1.3%	6.2%	**3.6%**	0.6%	7.0%	**3.7%**
	TCB Creation + Resync/Desync	78.1%	95.6%	**84.6%**	2.4%	18.6%	**12.9%**	0.9%	4.0%	**2.6%**
	TCB Teardown + TCB Reversal	84.6%	93.1%	**89.5%**	5.5%	8.7%	**7.1%**	0.1%	7.9%	**3.3%**

Table 4: Success rate of new strategies

evolved GFW devices to "re-transition" into the resynchronization state.

We combine the *TCB Reversal* strategy with the *TCB Teardown with RST* strategy. Specifically, as shown in Fig. 4, we first send a fake SYN/ACK packet from the client to the server to create a false TCB on the evolved GFW device. Next, we establish the legitimate 3-way handshake, which invalid with respect to the evolved GFW due to the existing TCB. Then we send a RST insertion packet to teardown the TCB on the old GFW model, followed by the HTTP request.

Avoiding interference from middleboxes or server. When crafting "insertion" packet, we choose the insertion packets wisely so as to not experience interference from the middleboxes, and not result in side-effects on the server. We primarily use TTL-based insertion packets since it is generally applicable. The key challenge here is to choose an accurate TTL value to hit the GFW, while not hitting server-side middleboxes or servers. We do that by first measuring the hop count from the client to the server using a way similar as *tcptraceroute*. Then, we subtract a small δ from the measured hop count, to try and prevent the insertion packet from reaching (hitting) the server-side middleboxes or the server. In our evaluation, we heuristically choose $\delta = 2$, but INTANG can iteratively change this to converge to a good value.

In addition, we exploit the new MD5 and old timestamp insertion packets, which allow the bypassing of the GFW without interfering with middelboxes or the server. Table 5 summarizes how we choose insertion packets for each type of TCP packet.

Packet Type	TTL	MD5	Bad ACK	Timestamp
SYN	✓			
RST	✓	✓		
Data	✓	✓	✓	✓

Table 5: Preferred construction of insertion packets

Results. We first analyze the results for individual evasion strategies. As seen from Table 4, the overall "Failure 2" rate is as low as 1% for all the strategies, which (a) show that our new strategies have a high success rate on the GFW which suggests that (b) our hypotheses with regards to the GFW evolution seem accurate.

We find that both the Failures 1 and Failures 2 always happen with regards to a few specific websites/IPs. One can presume that this is caused by some unknown GFW behavior or middlebox interference. However, since these cases are not sustained (are very rare), we argue that this is more likely to be due to middlebox interference.

Overall, we find that high Failure 1 rates is the major reason for overall low success rates. An introspective look suggests that because some servers/middleboxes accept packets regardless of the (wrong) ACK number or the presence of the MD5 option header, Failures 1 happen. Further, the TTL chosen is sometimes inaccurate due to (a) network dynamics or (b) hitting server-side middleboxes; this results in undesired side-effects that increase "Failures 1".

In addition, we find that for vantage points outside China, the TTL discrepancy unfortunately has a significant drawback. When accessing the servers in China, the GFW devices and the desired servers are usually within a few hops of each other (sometimes co-located). As a result it is extremely hard to converge to a TTL value for the insertion packet, that satisfies the requirement of hitting the GFW but not the server. As a consequence, in these scenarios, use of this discrepancy can cause either type of failures. We see from Table 4 that both the Failure 1 and Failure 2 rates are on average a bit higher than for the vantage points inside China.

Finally, because INTANG can choose the best strategy and insertion packets for each server IP based on historic results, we also evaluated INTANG performance in an additional row in Table 4 for vantage points inside China. It shows an average success rate of 98.3% which represents the performance with the optimal strategy specific to each website and network path. This is without further optimizing our implementation (*e.g.,* measuring packet losses and adjusting the level of redundancy for insertion packets).

Take away: While we do magnify the causes for failures, the biggest take away from this section is that our new hypothesized behaviors of the GFW seem to be fairly accurate, and that the new strategies are seemingly very effective in realizing the goal of evading the GFW, especially when the best strategies are chosen according to websites and network paths.

7.2 Evading TCP DNS Censorship

The GFW censors UDP DNS requests with DNS poisoning. It censors TCP DNS requests by injecting RST packets just like how it censors HTTP connections. Thus, our evasion strategies can also be used to help evade TCP DNS censorship. As discussed in §6, INTANG converts UDP DNS requests into TCP DNS requests. To

DNS resolver	IP	except Tianjin	All
Dyn 1	216.146.35.35	98.6%	92.7%
Dyn 2	216.146.36.36	99.6%	93.1%

Table 6: Success rate of TCP DNS censorship evasion

evaluate the effectiveness of our strategies on evading TCP DNS censorship, we use 2 public DNS resolvers from Dyn, and the same 11 vantage points in China. Google's DNS resolvers 8.8.8.8 and 8.8.4.4 have been IP hijacked by the GFW and thus cannot be used. By repeatedly requesting a censored domain, (*e.g.,* www.dropbox.com) 100 times, using the "improved TCB Teardown with RST strategy," we get the results shown in Table 6. The vantage points in Tianjin have low success rates of 38% and 24%. However, the others jointly yield success rates of over 99.5 %. Interestingly, we accidentally discover that if we use the TCP DNS through the two OpenDNS's DNS resolvers 208.67.222.222 and 208.67.220.220, even without applying INTANG we do not experience any censorship from any of our vantage points.

7.3 Tor and VPN

Tor is famous for supporting anonymous communications [22], and poses a serious threat to censorship. It is not surprising that it is reported that the GFW has been blocking Tor Bridge nodes through passive traffic analysis and active probing for more than 7 years [28]. Next, we examine if INTANG can help cover up Tor connections.

In our experiments, we first verify whether and how Tor nodes are blocked by the GFW. Subsequently, we test if INTANG can help clients from China evade Tor censorship.

We try to access hidden Tor bridge nodes setup on Amazon EC2 in the US from the same 11 vantage points (over 9 cities) (See § 3) inside China acting as Tor clients. Surprisingly, we find that there are four vantage points (in three cities Beijing, Zhangjiakou, and Qingdao) from which Tor connections to the hidden Tor bridge can operate without issues (as is) for over 2 days with periodic, manually generated traffic. Meanwhile, any hidden bridge nodes requested by the remaining 7 vantage points triggers active probing [13, 31] and are immediately blocked by the GFW, *i.e.,* any node in China can no longer connect to this IP via any port. This is very different from what was previously reported *i.e.,* the GFW only blocks the Tor port on that hidden bridge [31], and could cause collateral damage as the Amazon EC2 IPs are recycled. We test 5 different hidden bridge IPs and find no exceptions so far. The common characteristic of the first four locations is that they are all in Northern China. Thus, we speculate that Tor-filtering GFW nodes are most probably not encountered on the paths from this region.

Now, for the remaining vantage points where the Tor connections do trigger censorship blocking, we apply INTANG with the "improved TCB teardown strategy," five times each, and the success rate for the Tor connections is 100 %. We periodically repeat these experiments over a 9-hour period, and are able to keep using the Tor bridge node. This shows that: (a) our hypothesis that some of the GFW devices have evolved to a new model holds; and (b) INTANG is extremely effective in crafting the right measurement-driven strategy towards evading the GFW. We envision that Tor clients can even integrate INTANG in the future to improve its censorship evasion chances.

Similar to Tor, virtual private networks (VPN), which help users evade censorship, are also popular targets of the GFW. It is shown that there are multiple approaches used by the GFW to disconnect VPNs [30, 33]. They include DPI, IP address blocking, bandwidth throttling, *etc.* In November 2016, we set up an openvpn server in China, and used one node in America as a client. As per our experimental results, *a preliminary version of* INTANG helped openvpn over TCP evade censorship, while openvpn without INTANG was disconnected due to the client receiving a reset packet from the GFW during the handshake phase (the GFW seemingly used DPI). Unfortunately, we could not replay such experiments recently via either the PPTP protocol or with the openvpn protocol. Both protocols did not experience disconnections because of the GFW, nor did they suffer from rate limits imposed by the GFW. Unfortunately, we do not yet know what caused this change in behavior and we plan to continue monitoring the potential opportunity of applying INTANG to improve VPN stability.

8 DISCUSSION

GFW Countermeasures. Our work is based on the latest developments of the GFW. It is certainly possible that GFW may undergo additional improvements to defeat our evasion strategies, and we acknowledge that it is an arms race. For instance, we demonstrate that GFW is more liberal in accepting RST packets than normal servers. It is possible that the censor may perform additional checks on the RST packets (*e.g.,* checksum and MD5 option fields) as a defense. But that may open up a new evasion attack on the GFW (*e.g.,* when the server does not check MD5 option fields). One can also leverage GFW's agnostic nature to network topology. For example, we can measure the exact TTL value to bypass the GFW while not to reach the server (although it is also a challenge to achieve accuracy and efficiency simultaneously).

Another potential improvement the GFW can make is to trust the data packet sent by the client only after seeing the server's ACK packet acknowledging the appropriate sequence number. However, this will greatly complicates the GFW's design and implementation.

In summary, we believe this is an arms race. As GFW evolves, so can the evasion strategies. We believe that the cost of rolling out new GFW models is quite high and such evolution will happen at the timescale of months (if not years), which leaves enough time for evasion strategy development (especially when tools like INTANG are leveraged). For instance, as soon as the GFW evolves, a new GFW model will be derived and subjected to the "ignore path" analysis, which can lead to the generation of new evasion strategies.

Complexity and (sometimes) inconsistency of the GFW. During our long-term study of the GFW since 2015, we have observed that type-1 and type-2 resets sometimes occur individually. For example, on certain days, from a vantage point in CERNET Beijing we could observe only type-1 resets, while on other days, both types were seen. Our observations indicate that the two types of GFW devices are usually deployed together, and sometimes one is down. Also, we found there were some rather intricate effects when the two types were working together. During a measurement in May

2016, we found the type-1 GFW devices also have a subsequent 90-second blocking period (which it normally does not) as the type-2 devices does, after we using our new strategy to evade type-2 devices. And when we used no strategies, only the type-2 reset can be observed (*i.e.,* type-1 devices are not enforcing the 90-second blocking period). It looked like the type-2 reset suppressed type-1 reset. This rare behavior is not observed during other measurements. Furthermore, in May 2016 and May 2017, we have observed that RST packets sometimes were unable to tear down the TCB on the GFW, with different pairs of controlled clients and servers. This inconsistent behavior could be due to load balancing among different versions of the GFW, or some intricate effects caused by several GFW devices deployed together. However, we have no way to obtain the ground truth. We acknowledge our measurements are largely limited by being agnostic to the interference among different versions of GFW devices (or even middleboxes) and to the way how they are deployed, in addition to the blackbox nature of the GFW device itself. We are interested in further exploring this complexity and inconsistency in our future work.

Combination of Strategies. The GFW is heterogeneous with different co-existing versions. As a result, as we did in this paper, it is necessary to combine strategies that are effective against different versions of the GFW. This is normally not an issue as long as the strategies are not in conflict with each other. However, it is likely that the "Failure 1" rate will increase when a plurality of strategies are employed. This is because of the increase in insertion packets, which increases the likelihood of middlebox interference or side-effects on the server.

Ethical Considerations. All our experiments are carefully designed so as to not cause disruption to normal network operations. All connections are established from machines that we rent or control directly. The additional insertion packets are simply regular TCP packets (sometimes with incorrect field values) and may simply be discarded by the server. We control the traffic to each website to be low to avoid any unintended denial-of-service.

Note that INTANG doesn't guarentee unobservability for all its strategies. It is the user's discretion as to whether to use INTANG within the censor's jurisdiction. However, in China, due to heavy censorship [16], "crossing the wall" and accessing websites such as Google, Facebook, *etc.* has become a prevalent need. The censor usually punishes those who provide censorship circumvention services to the masses (*e.g.,* proxy/VPN providers) instead of punishing the users of the service. A client-side only tool like INTANG will be harder for the censor to trace and thwart.

9 RELATED WORK

We have already alluded to various related efforts throughout the paper (especially in § 2). They all focus on evaluating the censorship techniques *or* anti-censorship techniques aided by additional facilities, like VPN.

Clayton et al., propose to ignore the RST packets sent by the GFW [10]. This requires cooperation from the server-side, and is thus impractical (all servers will need to install a patch to do that). It does not prevent the censor from monitoring user traffic. Thus, we do not explicitly consider this in our work. As discussed earlier, Ptacek et al. [23], develop a deep understanding of the vulnerabilities of current NIDS, which has largely influenced later efforts (including ours) on TCP reset attack evasion. The West Chamber Project [25] is a censorship-circumvention tool that implemented the Ptacek et al.'s theory. However, it just uses two kinds of crafted packets to teardown the TCB on the GFW from both directions, and has now become ineffective.

Khattak et al.'s research [17] is the most relevant work to ours. Their strategies, and the problems thereof were already discussed in § 3. In addition, our measurement utilizes multiple vantage points instead of one vantage point as in [17]. Our measurement study leads to the discovery of the differences in deployment and features of the GFW from what was presented in that work. Li et al. [18] tested known TCP/IP insertion packets against censorship firewalls and DPI boxes in three countries and evaluated their effectiveness. In contrast, our work focuses on understanding and uncovering the latest development (new state machine) of the largest and most complex censorship system, which allows us to devise new evasion strategies.

10 CONCLUSION

In this paper we undertake, arguably, the most in depth measurement study of stateful (TCP-level) Internet censorship evasion on the GFW of China. Our work is divided into multiple stages. First, we perform extensive measurements of prior approaches and find that they are no longer effective. We attribute the reasons for this to two primary causes: (a) the GFW has evolved to imbibe new behaviors and, (b) the presence of middleboxes on the path between the client and the server that can interfere with the evasion strategies. Second, based on the knowledge gained, we hypothesize about new GFW behaviors and design new strategies that can possibly evade GFW today. We also build a novel, measurement driven tool INTANG that can converge on the right evasion strategy for a given client server pair. In the final stage, we perform extensive measurements of our new strategies and INTANG, and demonstrate that they provide near-to-perfect evasion rates when combined, thereby validating our new understanding of the GFW's stateful censorship model of today.

ACKNOWLEDGMENTS

This work was supported by Army Research Office under Grant No. 62954CSREP and the National Science Foundation under Grant No. 1464410, 1652954, and 1652954. We thank our shepherd Prof. Alan Mislove and the anonymous reviewers for their constructive comments towards improving the paper.

REFERENCES

[1] Giuseppe Aceto and Antonio Pescapé. 2015. Internet Censorship detection: A survey. *Computer Networks* 83, C, 381–421. https://doi.org/10.1016/j.comnet.2015.03.008

[2] Daniel Anderson. 2012. Splinternet Behind the Great Firewall of China. *Queue* 10, 11, Article 40, 10 pages. https://doi.org/10.1145/2390756.2405036

[3] Anonymous. 2009. Evaluation and Problems of Intrusion Detection System. (2009). Retrieved August 7, 2017 from http://www.chinagfw.org/2009/09/gfw_21.html

[4] Anonymous. 2012. The Collateral Damage of Internet Censorship by DNS Injection. *ACM SIGCOMM Computer Communication Review* 42, 3, 21–27. https://doi.org/10.1145/2317307.2317311

[5] Anonymous. 2014. Towards a Comprehensive Picture of the Great Firewall's DNS Censorship. In *4th USENIX Workshop on Free and Open Communications on*

the Internet (FOCI '14). USENIX Association, San Diego, CA. https://www.usenix.org/conference/foci14/workshop-program/presentation/anonymous

[6] Pablo Neira Ayuso. [n. d.]. Netfilter Queue Project. ([n. d.]). Retrieved August 7, 2017 from http://www.netfilter.org/projects/libnetfilter_queue/

[7] Yue Cao, Zhiyun Qian, Zhongjie Wang, Tuan Dao, Srikanth V. Krishnamurthy, and Lisa M. Marvel. 2016. Off-Path TCP Exploits: Global Rate Limit Considered Dangerous. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 209–225. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/cao

[8] Qi Alfred Chen, Zhiyun Qian, Yunhan Jack Jia, Yuru Shao, and Zhuoqing Morley Mao. 2015. Static Detection of Packet Injection Vulnerabilities: A Case for Identifying Attacker-controlled Implicit Information Leaks. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. ACM, New York, NY, USA, 388–400. https://doi.org/10.1145/2810103.2813643

[9] Vitaly Chipounov, Volodymyr Kuznetsov, and George Candea. 2012. The S2E Platform: Design, Implementation, and Applications. *ACM Transactions on Computer Systems (TOCS)* 30, 1, Article 2, 49 pages. https://doi.org/10.1145/2110356.2110358

[10] Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson. 2006. Ignoring the Great Firewall of China. In *Proceedings of the 6th International Conference on Privacy Enhancing Technologies (PET '06)*. Springer-Verlag, Berlin, Heidelberg, 20–35. https://doi.org/10.1007/11957454_2

[11] Jedidiah R. Crandall, Daniel Zinn, Michael Byrd, Earl Barr, and Rich East. 2007. ConceptDoppler: A Weather Tracker for Internet Censorship. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*. ACM, New York, NY, USA, 352–365. https://doi.org/10.1145/1315245.1315290

[12] Haixin Duan, Nicholas Weaver, Zongxu Zhao, Meng Hu, Jinjin Liang, Jian Jiang, Kang Li, and Vern Paxson. 2012. Hold-on: Protecting against on-path DNS poisoning. In *Workshop on Securing and Trusting Internet Names (SATIN)*.

[13] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. 2015. Examining How the Great Firewall Discovers Hidden Circumvention Servers. In *Proceedings of the 2015 Internet Measurement Conference (IMC '15)*. ACM, New York, NY, USA, 445–458. https://doi.org/10.1145/2815675.2815690

[14] Phillipa Gill, Masashi Crete-Nishihata, Jakub Dalek, Sharon Goldberg, Adam Senft, and Greg Wiseman. 2015. Characterizing Web Censorship Worldwide: Another Look at the OpenNet Initiative Data. *ACM Transactions on the Web (TWEB)* 9, 1, Article 4, 29 pages. https://doi.org/10.1145/2700339

[15] Andy Heffernan. 1998. *Protection of BGP Sessions via the TCP MD5 Signature Option*. RFC 2385. https://tools.ietf.org/html/rfc2385

[16] OpenNet Initiative. 2012. China | ONI Country Profile. (2012). Retrieved August 7, 2017 from https://opennet.net/research/profiles/china

[17] Sheharbano Khattak, Mobin Javed, Philip D. Anderson, and Vern Paxson. 2013. Towards Illuminating a Censorship Monitor's Model to Facilitate Evasion. In *Presented as part of the 3rd USENIX Workshop on Free and Open Communications on the Internet (FOCI '13)*. USENIX, Washington, D.C. https://www.usenix.org/conference/foci13/workshop-program/presentation/Khattak

[18] Fangfan Li, Abbas Razaghpanah, Arash Molavi Kakhki, Arian Akhavan Niaki, David Choffnes, Phillipa Gill, and Alan Mislove. 2017. lib•erate,(n): A library for exposing (traffic-classification) rules and avoiding them efficiently. In *Proceedings of the 2017 Internet Measurement Conference (IMC '17)*. ACM, London, UK. https://doi.org/10.1145/3131365.3131376

[19] Graham Lowe, Patrick Winters, and Michael L Marcus. 2007. *The Great DNS Wall of China*. Technical Report. https://censorbib.nymity.ch/pdf/Lowe2007a.pdf

[20] Jong Chun Park and Jedidiah R. Crandall. 2010. Empirical Study of a National-Scale Distributed Intrusion Detection System: Backbone-Level Filtering of HTML Responses in China. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems (ICDCS '10)*. IEEE Computer Society, Washington, DC, USA, 315–326. https://doi.org/10.1109/ICDCS.2010.46

[21] Jon Postel. 1981. *Transmission Control Protocol*. RFC 793. https://tools.ietf.org/html/rfc793

[22] The Tor Project. [n. d.]. The Tor Project. ([n. d.]). Retrieved August 7, 2017 from https://www.torproject.org

[23] Thomas H. Ptacek and Timothy N. Newsham. 1998. *Insertion, Envasion, and Denial of Service: Eluding Network Intrusion Detection*. Technical Report. http://www.icir.org/vern/Ptacek-Newsham-Evasion-98.ps

[24] Redis. [n. d.]. The Redis Project. ([n. d.]). Retrieved August 7, 2017 from http://redis.io/

[25] scholarzhang. 2010. West Chamber Project. (2010). Retrieved August 7, 2017 from https://code.google.com/p/scholarzhang/

[26] Zain Shamsi, Ankur Nandwani, Derek Leonard, and Dmitri Loguinov. 2014. Hershel: Single-packet Os Fingerprinting. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '14)*. ACM, New York, NY, USA, 195–206. https://doi.org/10.1145/2591971.2591972

[27] Michael Carl Tschantz, Sadia Afroz, David Fifield, and Vern Paxson. 2016. SoK: Towards Grounding Censorship Circumvention in Empiricism. In *2016 IEEE Symposium on Security and Privacy (SP)*. 914–933. https://doi.org/10.1109/SP.2016.59

[28] twilde. 2012. Knock Knock Knockin' on Bridges' Doors. (January 2012). Retrieved August 7, 2017 from https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors

[29] John-Paul Verkamp and Minaxi Gupta. 2012. Inferring Mechanics of Web Censorship Around the World. In *Presented as part of the 2nd USENIX Workshop on Free and Open Communications on the Internet (FOCI '12)*. USENIX, Bellevue, WA. https://www.usenix.org/conference/foci12/workshop-program/presentation/Verkamp

[30] VPNanswers.com. 2015. Bypass The Great Firewall And Hide Your OpenVPN In China. (2015). Retrieved August 7, 2017 from https://www.vpnanswers.com/bypass-great-firewall-hide-openvpn-in-china-2015/

[31] Philipp Winter and Stefan Lindskog. 2012. How the Great Firewall of China is Blocking Tor. In *Presented as part of the 2nd USENIX Workshop on Free and Open Communications on the Internet (FOCI '12)*. USENIX, Bellevue, WA. https://www.usenix.org/conference/foci12/workshop-program/presentation/Winter

[32] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. 2011. Telex: Anticensorship in the Network Infrastructure. In *Proceedings of the 20th USENIX Conference on Security (SEC '11)*. USENIX Association, Berkeley, CA, USA, 30–30. http://dl.acm.org/citation.cfm?id=2028067.2028097

[33] Eva Xiao. 2016. Behind The Scenes: Here's Why Your VPN Is Done In China. (2016). Retrieved August 7, 2017 from http://technode.com/2016/03/17/behind-scenes-heres-vpn/

[34] Xueyang Xu, Z. Morley Mao, and J. Alex Halderman. 2011. Internet Censorship in China: Where Does the Filtering Occur?. In *Proceedings of the 12th International Conference on Passive and Active Measurement (PAM '11)*. Springer-Verlag, Berlin, Heidelberg, 133–142. http://dl.acm.org/citation.cfm?id=1987510.1987524