Threat modeling and circumvention of Internet censorship

By

David Fifield

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor J.D. Tygar, Chair
Professor Deirdre Mulligan
Professor Vern Paxson

Fall 2017

Abstract

Threat modeling and circumvention of Internet censorship

by

David Fifield

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor J.D. Tygar, Chair

Research on Internet censorship is hampered by poor models of censor behavior. Censor models guide the development of circumvention systems, so it is important to get them right. A censor model should be understood not just as a set of capabilities—such as the ability to monitor network traffic—but as a set of priorities constrained by resource limitations.

My research addresses the twin themes of modeling and circumvention. With a grounding in empirical research, I build up an abstract model of the circumvention problem and examine how to adapt it to concrete censorship challenges. I describe the results of experiments on censors that probe their strengths and weaknesses; specifically, on the subject of active probing to discover proxy servers, and on delays in their reaction to changes in circumvention. I present two circumvention designs: domain fronting, which derives its resistance to blocking from the censor's reluctance to block other useful services; and Snowflake, based on quickly changing peer-to-peer proxy servers. I hope to change the perception that the circumvention problem is a cat-and-mouse game that affords only incremental and temporary advancements. Rather, let us state the assumptions about censor behavior atop which we build circumvention designs, and let those assumptions be based on an informed understanding of censor behavior.

Contents

Acknowledgements I wish to express special appreciation to those who have guided me: my advisor Doug Tygar, who helped me out of a tough spot; Vern Paxson; the other two caballeros, Sadia Afroz and Michael Tschantz; Xiao Qiang; Dan Boneh; and Steve Beaty.
I am grateful to those who offered me kindness or assistance in the course of my research: Angie Abbatecola; Barbara Goto; Nick Hopper; Lena Lau-Stewart; Heather Levien; Gordon Lyon; Deirdre Mulligan; Audrey Sillers; David Wagner; Philipp Winter, whose CensorBib is an invaluable resource; the Tor Project and the tor-dev and tor-qa mailing lists; OONI; the traffic-obf mailing list; the Open Technology Fund and the Freedom2Connect Foundation; and the SecML, BLUES, and censorship research groups at UC Berkeley. Thank you.
The opinions expressed herein are solely those of the author and do not necessarily represent any other person or organization.

Anti-acknowledgement My thesis is dedicated in opposition to the California State loyalty oath, a shameful relict of an un-American era.

Availability Source code and information related to this document are available at https://www.bamsoftware.com/papers/thesis/.

Chapter 1

Introduction

This is a thesis about Internet censorship. Specifically, it is about two threads of research that have occupied my attention for the past several years: gaining a better understanding of how censors work, and fielding systems that circumvent their restrictions. These two topics drive each other: better understanding leads to better circumvention systems that take into account censors' strengths and weaknesses; and the deployment of circumvention systems affords an opportunity to observe how censors react to changing circumstances. The output of my research takes two forms: models that describe how censors behave today and how they may evolve in the future, and tools for circumvention that are sound in theory and effective in practice.

1.1 Scope

Censorship is a big topic, and even adding the "Internet" qualifier makes it hardly less so. In order to deal with the subject in detail, I must limit the scope. The subject of this work is an important special case of censorship, which I call the "border firewall." See Figure 1.1.

A *client* resides within a network that is entirely controlled by a *censor*. Within the controlled network, the censor may observe, modify, inject, or block any communication along

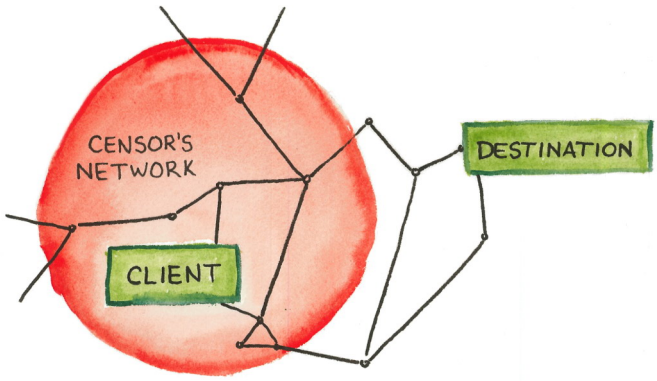

Figure 1.1: In the border firewall scenario, a client within a censor-controlled network wants to reach a destination on the outside.

any link. The client's computer, however, is trustworthy and not controlled by the censor. The censor tries to prevent some subset of the client's communication with the wider Internet, for instance by blocking those that discuss certain topics, that are destined to certain network addresses, or that use certain protocols. The client's goal is to evade the censor's controls and communicate with some *destination* that lies outside the censor's network; successfully doing so is called *circumvention*. Circumvention means somehow safely traversing a hostile network, eluding detection and blocking. The censor does not control the network outside its border; it may send messages to the outside world, but it cannot control them after they have traversed the border.

This abstract model is a good starting point, but it is not the whole story. We will have to adapt it to fit different situations, sometimes relaxing and sometimes strengthening assumptions. For example, the censor may be weaker than assumed: it may observe only the links that cross the border, not those that lie wholly inside; it may not be able to fully inspect every packet; or there may be deficiencies or dysfunctions in its detection capabilities. Or the censor may be stronger: while not fully controlling outside networks, it may perhaps exert outside influence to discourage network operators from assisting in circumvention. The client may be limited, for technical or social reasons, in the software and hardware they can use. The destination may knowingly cooperate with the client's circumvention effort, or may not. There are many possible complications, reflecting the messiness and diversity of dealing with real censors. Adjusting the basic model to reflect real-world actors' motivations and capabilities is the heart of *threat modeling*. In particular, what makes circumvention possible at all is the censor's motivation to block only some, but not all, of the incoming and outgoing communications—this assumption will be a major focus of the next chapter.

It is not hard to see how the border firewall model relates to censorship in practice. In a common case, the censor is the government of a country, and the limits of its controlled network correspond to the country's borders. A government typically has the power to enforce laws and control network infrastructure inside its borders, but not outside. However this is not the only case: the boundaries of censorship do not always correspond to the border of a country. Content restrictions may vary across geographic locations, even within the same country—Wright et al. [202] identified some reasons why this might be. A good model for some places is not a single unified regime, but rather several autonomous service providers, each controlling and censoring its own portion of the network, perhaps coordinating with others about what to block and perhaps not. Another important case is that of a university or corporate network, in which the only outside network access is through a single gateway router, which tries to enforce a policy on what is acceptable and what is not. These smaller networks often differ from national- or ISP-level networks in interesting ways, for instance with regard to the amount of overblocking they are willing to tolerate, or the amount of computation they can afford to spend on each communication.

Here are examples of forms of censorship that are in scope:

- blocking IP addresses
- blocking specific network protocols
- blocking DNS resolution for certain domains

- blocking keywords in URLs
- parsing application-layer data ("deep packet inspection")
- statistical and probabilistic traffic classification
- bandwidth throttling
- active scanning to discover the use of circumvention

Some other censorship-related topics that are *not* in scope include:

- domain takedowns (affecting all clients globally)
- server-side blocking (servers that refuse to serve certain clients)
- forum moderation and deletion of social media posts
- anything that takes place entirely within the censor's network and does not cross the border
- deletion-resistant publishing in the vein of the Eternity Service [10] (what Köpsell and Hillig call "censorship resistant publishing systems" [118 §1]), except insofar as access to such services may be blocked

Parts of the abstract model are deliberately left unspecified, to allow for the many variations that arise in practice. The precise nature of "blocking" can take many forms, from packet dropping, to injection of false responses, to softer forms of disruption such as bandwidth throttling. Detection does not have to be purely passive. The censor may to do work outside the context of a single connection; for example, it may compute aggregate statistics over many connections, make lists of suspected IP addresses, and defer some analysis for offline processing. The client may cooperate with other parties inside and outside the censor's network, and indeed almost all circumvention will require the assistance of a collaborator on the outside.

It is a fair criticism that the term "Internet censorship" in the title overreaches, given that I am talking only about one specific manifestation of censorship, albeit an important one. I am sympathetic to this view, and I acknowledge that far more topics could fit under the umbrella of Internet censorship. Nevertheless, for consistency and ease of exposition, in this document I will continue to use "Internet censorship" without further qualification to mean the border firewall case.

1.2 My background

This document describes my research experience from the past five years. The next chapter, "Principles of circumvention," is the thesis of the thesis, in which I lay out opinionated general principles of the field of circumvention. The remaining chapters are split between the topics of modeling and circumvention.

One's point of view is colored by experience. I will therefore briefly describe the background to my research. I owe much of my experience to collaboration with the Tor Project, producers of the Tor anonymity network. whose anonymity network has been the vehicle for deployment of my circumvention systems. Although Tor was not originally intended as a circumvention system, it has grown into one thanks to *pluggable transports*, a modularization system for circumvention implementations. I know a lot about Tor and pluggable transports, but I have less experience (especially implementation experience) with other systems, particularly those that are developed in languages other than English. And while I have plenty of operational experience—deploying and maintaining systems with real users—I have not been in a situation where I needed to circumvent regularly, as a user.

Chapter 2

Principles of circumvention

In order to understand the challenges of circumvention, it helps to put yourself in the mindset of a censor. A censor has two high-level functions: detection and blocking. Detection is a classification problem: the censor prefers to permit some communications and deny others, and so it must have some procedure for deciding which communications fall in which category. Blocking follows detection. Once the censor detects some prohibited communication, it must take some action to stop the communication, such as terminating the connection at a network router. Censorship requires both detection and blocking. (Detection without blocking would be called surveillance, not censorship.) The flip side of this statement is that circumvention has two ways to succeed: by eluding detection, or, once detected, by somehow resisting the censor's blocking action.

A censor is, then, essentially a traffic classifier coupled with a blocking mechanism. Though the design space is large, and many complications are possible, at its heart a censor must decide, for each communication, whether to block or allow, and then effect blocks as appropriate. Like any classifier, a censor is liable to make mistakes. When the censor fails to block something that it would have preferred to block, it is an error called a *false negative*; when the censor accidentally blocks something that it would have preferred to allow, it is a *false positive*. Techniques to avoiding detection are often called "obfuscation," and the term is an appropriate one. It reflects not an attitude of security through obscurity; but rather a recognition that avoiding detection is about making the censor's classification problem more difficult, and therefore more costly. Forcing the censor to trade false positives for false negatives is the core of all circumvention that is based on avoiding detection. The costs of misclassifications cannot be understood in absolute terms: they only have meaning relative to a specific censor and its resources and motivations. Understanding the relative importance that a censor assigns to classification errors—knowing what it prefers to allow and to block—is key to knowing what what kind of circumvention will be successful. Through good modeling, we can make the tradeoffs less favorable for the censor and more favorable for the circumventor.

The censor may base its classification decision on whatever criteria it finds practical. I like to divide detection techniques into two classes: *detection by content* and *detection by address*. Detection by content is based on the content or topic of the message: keyword filtering and protocol identification fall into this class. Detection by address is based on the sender or recipient of the message: IP address blacklists and DNS response tampering fall

into this class. An "address" may be any kind of identifier: an IP address, a domain name, an email address. Of these two classes, my experience is that detection by address is harder to defeat. The distinction is not perfectly clear because there is no clear separation between what is content and what is an address: the layered nature of network protocols means that one layer's address is another layer's content. Nevertheless, I find it useful to think about detection techniques in these terms.

The censor may block the address of the destination, preventing direct access. Any communication between the client and the destination must therefore be indirect. The indirect link between client and destination is called a *proxy*, and it must do two things: provide an unblocked address for the client to contact; and somehow mask the contents of the channel and the eventual destination address. I will use the word "proxy" expansively to encompass any kind of intermediary, not only a single host implementing a proxy protocol such an HTTP proxy or SOCKS proxy. A VPN (virtual private network) is also a kind of proxy, as is the Tor network, as may be a specially configured network router. A proxy is anything that acts on a client's behalf to assist in circumvention.

Proxies solve the first-order effects of censorship (detection by content and address), but they induce a second-order effect: the censor must now seek out and block proxies, in addition to the contents and addresses that are its primary targets. This is where circumvention research really begins: not with access to the destination per se, but with access to a proxy, which transitively gives access to the destination. The censor attempts to deal with detecting and blocking communication with proxies using the same tools it would for any other communication. Just as it may look for forbidden keywords in text, it may look for distinctive features of proxy protocols; just as it may block politically sensitive web sites, it may block the addresses of any proxies it can discover. The challenge for the circumventor is to use proxy addresses and proxy protocols that are difficult for the censor to detect or block.

The way of organizing censorship and circumvention techniques that I have presented is not the only one. Köpsell and Hillig [118 §4] divide detection into "content" and "circumstances"; their "circumstances" include addresses and also features that I consider more content-like: timing, data transfer characteristics, and protocols. Winter [198 §1.1] divides circumvention into three problems: bootstrapping, endpoint blocking, and traffic obfuscation. Endpoint blocking and traffic obfuscation correspond to my detection by address and detection by content; bootstrapping is the challenge of getting a copy of circumvention software and discovering initial proxy addresses. I tend to fold bootstrapping in with address-based detection; see Section 2.3. Khattak, Elahi, et al. break detection into four aspects [113 §2.4]: destinations, content, flow properties, and protocol semantics. I think of their "content," "flow properties," and "protocol semantics" as all fitting under the heading of content. My split between address and content mostly corresponds to Tschantz et al.'s "setup" and "usage" [182 §V] and Khattak, Elahi, et al.'s "communication establishment" and "conversation" [113 §3.1]. What I call "detection" and "blocking," Khattak, Elahi, et al. call "fingerprinting" and "direct censorship" [113 §2.3], and Tschantz et al. call "detection" and "action" [182 §II].

A major difficulty in developing circumvention systems is that however much you model and try to predict the reactions of a censor, real-world testing is expensive. If you really want to test a design against a censor, not only must you write and deploy an implementation, integrate it with client-facing software like web browsers, and work out details of its distribution—you

must also attract enough users to merit a censor's attention. Any system, even a fundamentally broken one, will work to circumvent most censors, as long as it is used only by one or only a few clients. The true test arises only after the system has begun to scale and the censor to fight back. This phenomenon may have contributed to the unfortunate characterization of censorship and circumvention as a cat-and-mouse game: deploying a flawed circumvention system, watching it become more popular and then get blocked, then starting over again with another similarly flawed system. In my opinion, the cat-and-mouse game is not inevitable, but is a consequence of inadequate understanding of censors. It is possible to develop systems that resist blocking—not absolutely, but quantifiably, in terms of costs to the censor—even after they have become popular.

2.1 Collateral damage

What prevents the censor from shutting down all connectivity within its network, trivially preventing the client from reaching any destination? The answer is that the censor derives benefits from allowing network connectivity, other than the communications which it wants to censor. Or to put it another way: the censor incurs a cost when it overblocks: accidentally blocks something it would have preferred to allow. Because it wants to block some things and allow others, the censor is forced to run as a classifier. In order to avoid harm to itself, the censor permits some measure of circumvention traffic.

The cost of false positives is of so central importance to circumvention that researchers have a special term for it: *collateral damage*. The term is a bit unfortunate, evoking as it does negative connotations from other contexts. It helps to focus more on the "collateral" than the "damage": collateral damage is any cost experienced *by the censor* as a result of incidental blocking done in the course of censorship. It must trade its desire to block forbidden communications against its desire to avoid harm to itself, balance underblocking with overblocking. Ideally, we force the censor into a dilemma: unable to distinguish between circumvention and other traffic, it must choose either to allow circumvention along with everything else, or else block everything and suffer maximum collateral damage. It is not necessary to reach this ideal fully before circumvention becomes possible. Better obfuscation drives up the censor's error rate and therefore the cost of any blocking. Ideally, the potential "damage" is never realized, because the censor sees the cost as being too great.

Collateral damage, being an abstract "cost," can take many forms. It may come in the form of civil discontent, as people try to access web sites and get annoyed with the government when unable to do so. It may be reduced productivity, as workers are unable to access resources they need to to their job. This is the usual explanation offered for why the Great Firewall of China has never blocked GitHub for for more than a few days, despite GitHub's being used to host and distribute circumvention software: GitHub is so deeply integrated into software development, that programmers cannot get work done when it is blocked.

Collateral damage, as with other aspects of censorship, cannot be understood in isolation, but only in relation to a particular censor. Suppose that blocking one web site results in the collateral blocking of a hundred more. Is that a large amount of collateral damage? It depends. Are those other sites likely to be visited by clients in the censor's network? Are

they in the local language? Do professionals and officials rely on them to get their job done? Is someone in the censorship bureau likely to get fired as a result of their blocking? If the answers to these question is yes, then yes, the collateral damage is likely to be high. But if not, then the censor could take or leave those hundred sites—it doesn't matter. Collateral damage is not just any harm that results from censorship, it is harm that is felt by the censor.

Censors may take actions to reduce collateral damage while still blocking most of what they intend to. (Another way to think of it is: reducing false positives without increasing false negatives.) For example, Winter and Lindskog [199], observed that the Great Firewall preferred to block individual ports, entire IP addresses, probably in a bid to reduce collateral damage. Local circumstances may serve to reduce collateral damage: for example if a domestic replacement exists for a foreign service, the censor may block the foreign service more easily.

The censor's reluctance to cause collateral damage is what makes circumvention possible in general. (There are some exceptions, discussed in the next section, where the censor can detect but for some reason cannot block.) To deploy a circumvention system is to make a bet: that the censor cannot field a classifier that adequately distinguishes the traffic of the circumvention system from other traffic which, if blocked, would result in collateral damage. Even steganographic circumvention channels that mimic some other protocol ultimately derive their blocking resistance from the potential of collateral damage. For example, a protocol that imitates HTTP can be blocked by blocking HTTP—the question then is whether the censor can afford to block HTTP. And that's in the best case, assuming that the circumvention protocol has no "tell" that enables the censor to distinguish it from the cover protocol it is trying to imitate. Indistinguishability is a necessary but not sufficient condition for blocking resistance: that which you are trying to be indistinguishable from must also have sufficient collateral damage. It's no use to have a perfect steganographic imitation of a protocol that the censor doesn't mind blocking.

In my opinion, collateral damage provides a more productive way to think about the behavior of censors than do alternatives. It takes into account different censors' differing resources and motivations, and so is more useful for generic modeling. Moreover, it gets to the heart of what makes traffic resistant to blocking. There are other ways of characterizing censorship resistance. Many authors—Burnett et al. [25], and Jones et al. Jones2014a, for instance—call the essential element "deniability," meaning that a client can plausibly claim to have been doing something other than circumventing when confronted with a log of their network activity. Khattak, Elahi, et al. [113 §4] consider "deniability" separately from "unblockability." Houmansadr et al. [103, 104, 105] used the term "unobservability," which I feel fails to capture the censor's essential function of distinguishing, not only observation. Brubaker et al. [23] used the term "entanglement," which I found enlightening. What they call entanglement I think of as indistinguishability—keeping in mind that that which you are trying to be indistinguishable from must be valued by the censor. Collateral damage provides a way to make statements about censorship resistance quantifiable, at least in a loose sense. Rather than saying, "the censor cannot block X," or even, "the censor is unwilling to block X," it is better to say "in order to block X, the censor would have to do Y," where Y is some action bearing a cost for the censor. A statement like this makes it clear that some censors may be able to afford the cost of blocking and others may not; there is no "unblockability" in absolute terms. Now, actually quantifying the value of Y is a task in itself, by no means a trivial one. A challenge for future work in this field is to assign

actual numbers (e.g., in dollars) to the costs borne by censors. If a circumvention system becomes blocked, it may simply mean that the circumventor overestimated the collateral damage or underestimated the censor's capacity to absorb it.

We have observed that the risk of collateral damage is what prevents the censor from shutting down the network completely—and yet, censors *do* occasionally enact shutdowns or daily "curfews." Shutdowns are costly—West [191] looked at 81 shutdowns in 19 countries in 2015 and 2016, and estimated that they collectively cost $2.4 billion in losses to gross domestic product. Deloitte [40] estimated that shutdowns cost millions of dollars per day per 10 million population, the amount depending on a country's level of connectivity. This does not necessarily contradict the theory of collateral damage. It is just that, in some cases, a censor reckons that the benefits of a shutdown outweigh the costs. As always, the outcome depends on the specific censor: censors that don't benefit as much from the Internet don't have as much to lose by blocking it. The fact that shutdowns are limited in duration shows that even censors that can afford to a shutdown cannot afford to keep it up forever.

Complicating everything is the fact that censors are not bound to act rationally. Like any other large, complex entity, a censor is prone to err, to act impetuously, to make decisions that cause more harm than good. The imposition of censorship in the first place, I suggest, is exactly such an irrational action, retarding progress at the greater societal level.

2.2 Content obfuscation strategies

There are two general strategies to counter content-based detection. The first is to mimic some content that the censor allows, like HTTP or email. The second is to randomize the content, making it dissimilar to anything that the censor specifically blocks.

Tschantz et al. [182] call these two strategies "steganography" and "polymorphism" respectively. It is not a strict categorization—any real system will incorporate a bit of both. The two strategies reflect they reflect differing conceptions of censors. Steganography works against a "whitelisting" or "default-deny" censor, one that permits only a set of specifically enumerated protocols and blocks all others. Polymorphism, on the other hand, fails against a whitelisting censor, but works against a "blacklisting" or "default-allow" censor, one that blocks a set of specifically enumerated protocols and allows all others.

This is not to say that steganography is strictly superior to polymorphism—there are tradeoffs in both directions. Effective mimicry can be difficult to achieve, and in any case its effectiveness can only be judged against a censor's sensitivity to collateral damage. Whitelisting, by its nature, tends to cause more collateral damage than blacklisting. And just as obfuscation protocols are not purely steganographic or polymorphic, real censors are not purely whitelisting or blacklisting. Houmansadr et al. [103] exhibited weaknesses in "parrot" circumvention systems that imperfectly mimic a cover protocol. Mimicking a protocol in every detail, down to its error behavior, is difficult, and any inconsistency is a potential feature that a censor may exploit. Wang et al. [186] found that some of the proposed attacks against parrot systems would be impractical due to high false-positive rates, but offered other attacks designed for efficiency and low false positives. Geddes et al. [95] showed that even perfect imitation may leave vulnerabilities due to mismatches between the cover protocol and the carried protocol. For instance, randomly dropping packets may disrupt circumvention

more than normal use of the cover protocol. It's worth noting, though, that apart from active probing and perhaps entropy measurement, most of the attacks proposed in academic research have not been used by censors in practice.

Some systematizations (for example those of Brubaker et al. [23 §6]; Wang et al. [186 §2]; and Khattak, Elahi, et al. [113 §6.1]) further subdivide steganographic systems into those based on mimicry (attempting to replicate the behavior of a cover protocol) and tunneling (sending through a genuine implementation of the cover protocol). I do not find the distinction very useful, except when discussing concrete implementation choices. To me, there is no clear division: there are various degrees of fidelity in imitation, and tunneling only tends to offer higher fidelity than does mimicry.

I will list some circumvention systems that represent the steganographic strategy. Infranet [62], way back in 2002, built a covert channel within HTTP, encoding upstream data as crafted requests and downstream data as steganographic images. StegoTorus [190] uses custom encoders to make traffic resemble common HTTP file types, such as PDF, JavaScript, and Flash. SkypeMorph [139] mimics a Skype video call. FreeWave [105] modulates a data stream into an acoustic signal and transmits it over VoIP. Format-transforming encryption, or FTE [58], force traffic to conform to a user-specified syntax: if you can describe it, you can imitate it. Despite receiving much research attention, steganographic systems have not been as used in practice as polymorphic ones. Of the listed systems, only FTE has seen substantial deployment.

There are many examples of the randomized, polymorphic strategy. An important subclass of these comprises the so-called look-like-nothing systems that encrypt a stream without any plaintext header or framing information, so that it appears to be a uniformly random byte sequence. A pioneering design was the obfuscated-openssh of Bruce Leidl [120], which aimed to hide the plaintext packet metadata in the SSH protocol. obfuscated-openssh worked, in essence, by first sending an encryption key, and then sending ciphertext encrypted with that key. The encryption of the obfuscation layer was an additional layer, independent of SSH's ordinary encryption. A censor could, in principle, passively detect and deobfuscate the protocol by recovering the key and using it to decrypt the rest of the stream. obfuscated-openssh could optionally incorporate a pre-shared password into the key derivation function, which would protect against this attack. Dust [195], similarly randomized bytes (at least in its v1 version—later versions permitted fitting to distributions other than uniform). It was not susceptible to passive deobfuscation because it required an out-of-band key exchange to happen before each session. Shadowsocks [170] is a lightweight encryption layer atop a simple proxy protocol.

There is a line of successive look-like-nothing protocols—obfs2, obfs3, ScrambleSuit, and obfs4—which I like because they illustrate the mutual advances of censors and circumventors over several years. obfs2 [110], which debuted in 2012 in response to blocking in Iran [43], uses very simple obfuscation inspired by obfuscated-openssh: it is essentially equivalent to sending an encryption key, then the rest of the stream encrypted with that key. obfs2 is detectable, with no false negatives and negligible false positives, by even a passive censor who knows how it works; and it is vulnerable to active probing attacks, where the censor speculatively connects to servers to see what protocols they use. However, it sufficed against the keyword- and pattern-based censors of its era. obfs3 [111]—first available in 2013 but not really released to users until 2014 [152]—was designed to fix the passive detectability of its predecessor.

obfs3 employs a Diffie–Hellman key exchange that prevents easy passive detection, but it can still be subverted by an active man in the middle, and remains vulnerable to active probing. (The Great Firewall of China had begun active-probing for obfs2 by January 2013, and for obfs3 by August 2013—see Table 4.2.) ScrambleSuit [200], first available to users in 2014 [29], arose in response to the active-probing of obfs3. Its innovations were the use of an out-of-band secret to authenticate clients, and traffic shaping techniques to perturb the underlying stream's statistical properties. When a client connects to a ScrambleSuit proxy, it must demonstrate knowledge of the out-of-band secret before the proxy will respond, which prevents active probing. obfs4 [206], first available in 2014 [154], is an incremental advancement on ScrambleSuit that uses more efficient cryptography, and additionally authenticates the key exchange to prevent active man-in-the-middle attacks.

There is an advantage in designing polymorphic protocols, as opposed to steganographic ones, which is that every proxy can potentially have its own characteristics. ScrambleSuit and obfs4, in addition to randomizing packet contents, also shape packet sizes and timing to fit random distributions. Crucially, the chosen distributions are consistent within each proxy, but vary across proxies. That means that even if a censor is able to build a profile for a particular proxy, it is not necessarily useful for detecting other instances.

2.3 Address blocking resistance strategies

The first-order solution for reaching a destination whose address is blocked is to instead route through a proxy. But a single, static proxy is not much better than direct access, for circumvention purposes—a censor can block the proxy just as easily as it can block the destination. Circumvention systems must come up with ways of addressing this problem.

There are two reasons why resistance to blocking by address is challenging. The first is due to the nature of network routing: the client must, somehow, encode the address of the destination into the messages it sends. The second is the insider attack: legitimate clients must have some way to discover the addresses of proxies. By pretending to be a legitimate client, the censor can learn those addresses in the same way.

Compared to content obfuscation, there are relatively few strategies for resistance to blocking by address. They are basically five:

- sharing private proxies among only a few clients
- having a large population of secret proxies and distributing them carefully
- having a very large population of proxies and treating them as disposable
- proxying through a service with high collateral damage
- address spoofing

The simplest proxy infrastructure is no infrastructure at all: require every client to set up and maintain a proxy for their own personal use, or for a few of their friends. As long as the use of any single address remains low, it may escape the censor's notice [49 §4.2]. The problem with this strategy, of course, is usability and scalability. If it were easy for everyone

to set up their own proxy on an unblocked address, they would do it, and blocking by address would not be a concern. The challenge is making such techniques general so they are usable by more than experts. uProxy [184] is now working on just that: automating the process of setting up a proxy on a server.

What Köpsell and Hillig call the "many access points" model [118 §5.2] has been adopted in some form by many circumvention systems. In this model, there are many proxies in operation. They may be full-fledged general-purpose proxies, or only simple forwarders to a more capable proxy. They may be operated by volunteers or coordinated centrally. In any case, the success of the system hinges on being able to sustain a population of proxies, and distribute information about them to legitimate users, without revealing too many to the censor. Both of these considerations pose challenges.

Tor's blocking resistance design [49], based on secret proxies called "bridges," was of this kind. Volunteers run bridges, which report themselves to a central database called BridgeDB [181]. Clients contact BridgeDB through some unblocked out-of-band channel (HTTPS, email, or word of mouth) in order to learn bridge addresses. The BridgeDB server takes steps to prevent the easy enumeration of its database [124]. Each request returns only a small set of bridges, and repeated requests by the same client return the same small set (keyed by a hash of the client's IP address prefix or email address). Requests through the HTTPS interface require the client to solve a captcha, and email requests are honored only from the domains of email providers that are known to limit the rate of account creation. The population of bridges is partitioned into "pools"—one pool for HTTPS distribution, one for email, and so on—so that if an adversary manages to enumerate one of the pools, it does not affect the bridges of the others. But even these defenses may not be enough. Despite public appeals for volunteers to run bridges (for example Dingledine's initial call in 2007 [44]), there have never been more than a few thousand of them, and Dingledine reported in 2011 that the Great Firewall of China managed to enumerate both the HTTPS and email pools [45 §1, 46 §1].

Tor relies on BridgeDB to provide address blocking resistance for all its transports that otherwise have only content obfuscation. And that is a great strength of such a system. It enables, to some extent, content obfuscation to be developed independently, and rely on an existing generic proxy distribution mechanism in order to produce an overall working system. There is a whole line of research, in fact, on the question of how best to distribute information about an existing population of proxies, which is known as the "proxy distribution problem" or "proxy discovery problem." Proposals such as Proximax [134], rBridge [188], and Salmon [54] aim to make proxy distribution robust by tracking the reputation of clients and the unblocked lifetimes of proxies.

A way to make proxy distribution more robust against censors (but at the same time less usable by clients) is to "poison" the set of proxy addresses with the addresses of important servers, the blocking of which would result in high collateral damage. VPN Gate employed this idea [144 §4.2], mixing into the their public proxy list the addresses of root DNS servers and Windows Update servers.

Apart from "in-band" discovery of bridges via subversion of a proxy distribution system, one must also worry about "out-of-band" discovery, for example by mass scanning [46 §6, 49 §9.3]. Durumeric et al. found about 80% of existing (unobfuscated) Tor bridges [57 §4.4] by scanning all of IPv4 on a handful of common bridge ports. Matic et al. had similar results

in 2017 [133 §V.D], using public search engines in lieu of active scanning. The best solution to the scanning problem is to do as ScrambleSuit [200], obfs4 [206], and Shadowsocks [170] do, and associate with each proxy a secret, without which a scanner cannot initiate a connection. Scanning for bridges is closely related to active probing, the topic of Chapter 4.

Another way of achieving address blocking resistance is to treat proxies as temporary and disposable, rather than permanent and valuable. This is the idea underlying flash proxy [84] and Snowflake (Chapter 7). Most proxy distribution strategies are designed around proxies lasting at least on the order days. In contrast, disposable proxies may last only minutes or hours. Setting up a Tor bridge or even something lighter-weight like a SOCKS proxy still requires installing some software on a server somewhere. The proxies of flash proxy and Snowflake have a low set-up and tear-down cost: you can run one just by visiting a web page. These designs do not need a sophisticated proxy distribution strategy as long as the rate of proxy creation is kept higher than the censor's rate of discovery.

The logic behind diffusing many proxies widely is that a censor would have to block large swaths of the Internet in order to effectively block them. However, it also makes sense to take the opposite tack: have just one or a few proxies, but choose them to have high enough collateral damage that the censor does not dare block them. Refraction networking [160] puts proxy capability into network routers—in the middle of paths, rather than at the end. Clients cryptographically tag certain flows in a way that is invisible to the censor but detectable to a refraction-capable router, which redirects from its apparent destination to some other, covert destination. In order to prevent circumvention, the censor has to induce routes that avoid the special routers [168], which is costly [106]. Domain fronting [89] has similar properties. Rather than a router, it uses another kind of network intermediary: a content delivery network. Using properties of HTTPS, a client may request one site while appearing (to the censor) to request another. Domain fronting is the topic of Chapter 6. The big advantage of this general strategy is that the proxies do not need to be kept secret from the censor.

The final strategy for address blocking resistance is address spoofing. The notable design in this category is CensorSpoofer [187]. A CensorSpoofer client never communicates directly with a proxy. It sends upstream data through a low-bandwidth, indirect channel such as email or instant messaging, and downstream data through a simulated VoIP conversation, spoofed to appear as if it were coming from some unrelated dummy IP address. The asymmetric design is feasible because of the nature of web browsing: typical clients send much less than they receive. The client never even needs to know the actual address of the proxy, meaning that CensorSpoofer has high resistance to insider attack: even running the same software as a legitimate client, the censor does not learn enough information to effect a block. The idea of address spoofing goes back farther; as early as 2001, TriangleBoy [167] employed lighter-weight intermediate proxies that simply forwarded client requests to a long-lived proxy at a static, easily blockable address. In the downstream direction, the long-lived proxy would, rather than route back through the intermediate proxy, only spoof its responses to look as if they came from proxy. TriangleBoy did not match CensorSpoofer's resistance to insider attack, because clients still needed to find and communicate directly with a proxy, so the whole system basically reduced to the proxy discovery problem, despite the use of address spoofing.

2.4 Spheres of influence and visibility

It is usual to assume, conservatively, that whatever the censor can detect, it also can block; that is, to ignore blocking per se and focus only on the detection problem. We know from experience, however, that there are cases in practice where a censor's reach exceeds its grasp: where it is able to detect circumvention but for some reason cannot block it. It may be useful to consider this possibility when modeling. Khattak, Elahi, et al. [113] express it nicely by subdividing the censor's network into a *sphere of influence* within which the censor has active control, and a potentially larger *sphere of visibility* within which the censor may only observe, but not act.

A landmark example of this kind of thinking is the 2006 research on "Ignoring the Great Firewall of China" by Clayton et al. [31]. They found that the firewall would block connections by injecting phony TCP RST packets (which cause the connection to be torn down) or SYN/ACK packets (which cause the connection to become unsynchronized), and that simply ignoring the anomalous packets rendered blocking ineffective. (Why did the censor choose to *inject* its own packets, rather than *drop* those of the client or server? The answer is probably that injection is technically easier to achieve, highlighting a limit on the censor's power.) One can think of this ignoring as shrinking the censor's sphere of influence: it can still technically act within this sphere, but not in a way that actually achieves blocking. Additionally, intensive measurements revealed many failures to block, and blocking rates that changed over time, suggesting that even when the firewall intends a policy of blocking, it does not always succeed.

Another fascinating example of "look, but don't touch" communication is the "filecasting" technique used by Toosheh [142], a file distribution service based on satellite television broadcasts. Clients tune their satellite receivers to a certain channel and record the broadcast to a USB flash drive. Later, they run a program on the recording that decodes the information and extracts a bundle of files. The system is unidirectional: clients can only receive the files that the operators choose to provide. The censor can easily see that Toosheh is in use—it's a broadcast, after all—but cannot identify users, or block the signal in any way short of continuous radio jamming or tearing down satellite dishes.

There are parallels between the study of Internet censorship and that of network intrusion detection. One is that a censor's detector may be implemented as a network intrusion detection system or monitor, a device "on the side" of a communication link that receives a copy of the packets that flow over the link, but that, unlike a router, is not responsible for forwarding the packets onward. Another parallel is that censors are susceptible to the same kinds of evasion and obfuscation attacks that affect network monitors more generally. In 1998, Ptacek and Newsham [158] and Paxson [149 §5.3] outlined various attacks against network intrusion detection systems—such as manipulating the IP time-to-live field or sending overlapping IP fragments—that cause a monitor either to accept what the receiver will reject, or reject what the receiver will accept. A basic problem is that a monitor's position in the middle of the network does not enable it to predict exactly how each packet will be interpreted by the endpoints. Cronin et al. [36] posit that the monitor's conflicting goals of sensitivity (recording all that is relevant) and selectivity (recording *only* what is relevant) give rise to an unavoidable "eavesdropper's dilemma."

Monitor evasion techniques can be used to reduce a censor's sphere of visibility—remove

certain traffic features from its consideration. Crandall et al. [33] in 2007 suggested using IP fragmentation to prevent keyword matching. In 2008 and 2009, Park and Crandall [148] explicitly characterized the Great Firewall as a network intrusion detection system and found that a lack of TCP reassembly allowed evading keyword matching. Winter and Lindskog [199] found that the Great Firewall still did not do TCP segment reassembly in 2012. They released a tool, brdgrd [196], that by manipulating the TCP window size, prevented the censor's scanners from receiving a full response in the first packet, thereby foiling active probing. Anderson [9] gave technical information on the implementation of the Great Firewall as it existed in 2012, and observed that it is implemented as an "on-the-side" monitor. Khattak et al. [114] applied a wide array of evasion experiments to the Great Firewall in 2013, identifying classes of working evasions and estimating the cost to counteract them. Wang et al. [189] did further evasion experiments against the Great Firewall a few years later, finding that the firewall had evolved to prevent some previous evasion techniques, and discovering new ones.

2.5 Early censorship and circumvention

Internet censorship and circumvention began to rise to importance in the mid-1990s, coinciding with the popularization of the World Wide Web. Even before national-level censorship by governments became an issue, researchers investigated the blocking policies of personal firewall products—those intended, for example, for parents to install on the family computer. Meeks and McCullagh [138] reported in 1996 on the secret blocking lists of several programs. Bennett Haselton and Peacefire [100] found many cases of programs blocking more than they claimed, including web sites related to politics and health.

Governments were not far behind in building legal and technical structures to control the flow of information on the web, in some cases adapting the same technology originally developed for personal firewalls. The term "Great Firewall of China" first appeared in an article in *Wired* [15] in 1997. In the wake of the first signs of blocking by ISPs, people were thinking about how to bypass filters. The circumvention systems of that era were largely HTML-rewriting web proxies: essentially a form on a web page into which a client would enter a URL. The server would fetch the desired page on behalf of the client, and before returning the response, rewrite all the links and external references in the page to make them relative to the proxy. CGIProxy [131], SafeWeb [132], Circumventor [99], and the first version of Psiphon [28] were all of this kind.

These systems were effective against their censors of their day—at least with respect to the blocking of destinations. They had the major advantage of requiring no special client-side software other than a web browser. The difficulty they faced was second-order blocking as censors discovered and blocked the proxies themselves. Circumvention designers deployed some countermeasures; for example Circumventor had a mailing list [49 §7.4] which would send out fresh proxy addresses every few days. A 1996 article by Rich Morin [140] presented a prototype HTML-rewriting proxy called Rover, which eventually became CGIProxy. The article predicted the failure of censorship based on URL or IP address, as long as a significant fraction of web servers ran such proxies. That vision has not come to pass. Accumulating a sufficient number of proxies and communicating their addresses securely to clients—in short,

the proxy distribution problem—turned out not to follow automatically, but to be a major sub-problem of its own.

Threat models had to evolve along with censor capabilities. The first censors would be considered weak by today's standards, mostly easy to circumvent by simple countermeasures, such as tweaking a protocol or using an alternative DNS server. (We see the same progression play out again when countries first begin to experiment with censorship, such as in Turkey in 2014, where alternative DNS servers briefly sufficed to circumvent a block of Twitter [35].) Not only censors were changing—the world around them was changing as well. In field of circumvention, which is so heavily affected by concerns about collateral damage, the milieu in which censors operate is as important as the censors themselves. A good example of this is the paper on Infranet, the first academic circumvention design I am aware of. Its authors argued, not unreasonably for 2001, that TLS would not suffice as a cover protocol [62 §3.2], because the relatively few TLS-using services at that time could all be blocked without much harm. Certainly the circumstances are different today—domain fronting and all refraction networking schemes require the censor to permit TLS. As long as circumvention remains relevant, it will have to change along with changing times, just as censors do.

Chapter 3

Understanding censors

The main tool we have to build relevant threat models is the study of censors. The study of censors is complicated by difficulty of access: censors are not forthcoming about their methods. Researchers are obligated to treat censors as a black box, drawing inferences about their internal workings from their externally visible characteristics. The easiest thing to learn is the censor's *what*—the destinations and contents that are blocked. Somewhat harder is the investigation into *where* and *how*, the specific technical mechanisms used to effect censorship and where they are deployed in the network. Most difficult to infer is the *why*, the motivations and goals that underlie an apparatus of censorship.

From past measurement studies we may draw a few general conclusions. Censors change over time, and not always in the direction of more restrictions. Censorship differs greatly across countries, not only in subject but in mechanism and motivation. However it is reasonable to assume a basic set of capabilities that many censors have in common:

- blocking of specific IP addresses and ports
- control of default DNS servers
- blocking DNS queries
- injection of false DNS responses
- injection of TCP RSTs
- keyword filtering in unencrypted contents
- application protocol parsing ("deep packet inspection")
- participation in a circumvention system as a client
- scanning to discover proxies
- throttling connections
- temporary total shutdowns

Not all censors will be able—or motivated—to do all of these. As the amount of traffic to be handled increases, in-path attacks such as throttling become relatively more expensive. Whether a particular act of censorship even makes sense will depend on a local cost–benefit analysis, a weighing of the expected gains against the potential collateral damage. Some censors may be able to tolerate a brief total shutdown, while for others the importance of Internet connectivity is too great for such a blunt instrument.

The Great Firewall of China (GFW), because of its unparalleled technical sophistication, is tempting to use as a model adversary. There has indeed been more research focused on China than any other country. But the GFW is in many ways an outlier, and not representative of other censors. A worldwide view is needed.

Building accurate models of censor behavior is not only needed for the purpose of circumvention. It also has implications for ethical measurement [202 §5, 108 §2]. For example, a common way to test for censorship is to ask volunteers to run software that connects to potentially censored destinations and records the results. This potentially puts volunteers at risk. Suppose the software accesses a destination that violates local law. Could the volunteer be held liable for the access? Quantifying the degree of risk depends on modeling how a censor will react to a given stimulus [32 §2.2].

3.1 Censorship measurement studies

A large part of research on censorship is composed of studies of censor behavior in the wild. In this section I summarize past studies, which, taken together, present a picture of censor behavior in general. They are based on those in an evaluation study done by me and others in 2016 [182 §IV.A]. The studies are diverse and there are many possible ways to categorize them. Here, I have divided them into one-time experiments and generic measurement platforms.

One-shot studies

One of the earliest technical studies of censorship occurred in a place you might not expect, the German state of North Rhein-Westphalia. Dornseif [52] tested ISPs' implementation of a controversial legal order to block web sites circa 2002. While there were many possible ways to implement the block, none were trivial to implement, nor free of overblocking side effects. The most popular implementation used DNS tampering, which is returning (or injecting) false responses to DNS requests for the blocked sites. An in-depth survey of DNS tampering found a variety of implementations, some blocking more and some blocking less than required by the order. This time period seems to mark the beginning of censorship by DNS tampering in general; Dong [51] reported it in China in late 2002.

Zittrain and Edelman [208] used open proxies to experimentally analyze censorship in China in late 2002. They tested around 200,000 web sites and found around 19,000 of them to be blocked. There were multiple methods of censorship: web server IP address blocking, DNS server IP address blocking, DNS poisoning, and keyword filtering.

Clayton [30] in 2006 studied a "hybrid" blocking system, CleanFeed by the British ISP BT, that aimed for a better balance of costs and benefits: a "fast path" IP address and port matcher acted as a prefilter for the "slow path," a full HTTP proxy. The system, in use

since 2004, was designed to block access to any of a secret list of web sites. The system was vulnerable to a number of evasions, such a using a proxy, using an alternate IP address or port, and obfuscating URLs. The two-level nature of the blocking system unintentionally made it an oracle that could reveal the IP addresses of sites in the secret blocking list.

In 2006, Clayton, Murdoch, and Watson [31] further studied the technical aspects of the Great Firewall of China. They relied on an observation that the firewall was symmetric, treating incoming and outgoing traffic equally. By sending web requests from outside the firewall to a web server inside, they could provoke the same blocking behavior that someone on the inside would see. They sent HTTP requests containing forbidden keywords that caused the firewall to inject RST packets towards both the client and server. Simply ignoring RST packets (on both ends) rendered the blocking mostly ineffective. The injected packets had inconsistent TTLs and other anomalies that enabled their identification. Rudimentary countermeasures, such as splitting keywords across packets, were also effective in avoiding blocking. The authors brought up an important point that would become a major theme of future censorship modeling: censors are forced to trade blocking effectiveness against performance. In order to cope with high load at a reasonable costs, censors may employ the "on-path" architecture of a network monitor or intrusion detection system; i.e., one that can passively monitor and inject packets, but cannot delay or drop them.

Contemporaneous studies of the Great Firewall by Wolfgarten [201] and Tokachu [175] found cases of DNS tampering, search engine filtering, and RST injection caused by keyword detection. In 2007, Lowe, Winters, and Marcus [125] did detailed experiments on DNS tampering in China. They tested about 1,600 recursive DNS servers in China against a list of about 950 likely-censored domains. For about 400 domains, responses came back with bogus IP addresses, chosen from a set of about 20 distinct IP addresses. Eight of the bogus addresses were used more than the others: a whois lookup placed them in Australia, Canada, China, Hong Kong, and the U.S. By manipulating the IP time-to-live field, the authors found that the false responses were injected by an intermediate router, evidenced by the fact that the authentic response would be received as well, only later. A more comprehensive survey [12] of DNS tampering occurred in 2014, giving remarkable insight into the internal structure of the censorship machines. DNS injection happened only at border routers. IP ID and TTL analysis showed that each node was a cluster of several hundred processes that collectively injected censored responses. They found 174 bogus IP addresses, more than previously documented, and extracted a blacklist of about 15,000 keywords.

The Great Firewall, because of its unusual sophistication, has been an enduring object of study. Part of what makes it interesting is its many blocking modalities, both active and passive, proactive and reactive. The ConceptDoppler project of Crandall et al. [33] measured keyword filtering by the Great Firewall and showed how to discover new keywords automatically by latent semantic analysis, using the Chinese-language Wikipedia as a corpus. They found limited statefulness in the firewall: sending a naked HTTP request without a preceding SYN resulted in no blocking. In 2008 and 2009, Park and Crandall [148] further tested keyword filtering of HTTP responses. Injecting RST packets into responses is more difficult than doing the same to requests, because of the greater uncertainty in predicting TCP sequence numbers once a session is well underway. In fact, RST injection into responses was hit or miss, succeeding only 51% of the time, with some, apparently diurnal, variation. They also found inconsistencies in the statefulness of the firewall. Two of ten injection

servers would react to a naked HTTP request; that it, one sent outside of an established TCP connection. The remaining eight of ten required an established TCP connection. Xu et al. [204] continued the theme of keyword filtering in 2011, with the goal of discovering where filters are located at the IP and autonomous system levels. Most filtering is done at border networks (autonomous systems with at least one peer outside China). In their measurements, the firewall was fully stateful: blocking was never triggered by an HTTP request outside an established TCP connection. Much filtering occurred at smaller regional providers, rather than on the network backbone. Anderson [9] gave a detailed description of the design of the Great Firewall in 2012. He described IP address blocking by null routing, RST injection, and DNS poisoning, and documented cases of collateral damage affecting clients inside and outside China.

Dainotti et al. [37] reported on the total Internet shutdowns that took place in Egypt and Libya in the early months of 2011. They used multiple measurements to document the outages as they occurred. During the shutdowns, they measured a drop in scanning traffic (mainly from the Conficker botnet). By comparing these different measurements, they showed that the shutdown in Libya was accomplished in more than one way, both by altering network routes and by firewalls dropping packets.

Winter and Lindskog [199], and later Ensafi et al. [60] did a formal investigation into active probing, a reported capability of the Great Firewall since around October 2011. They focused on the firewall's probing of Tor and its most common pluggable transports.

Anderson [6] documented network throttling in Iran, which occurred over two major periods between 2011 and 2012. Throttling degrades network access without totally blocking it, and is harder to detect than blocking. Academic institutions were affected by throttling, but less so than other networks. Aryan et al. [14] tested censorship in Iran during the two months before the June 2013 presidential election. They found multiple blocking methods: HTTP request keyword filtering, DNS tampering, and throttling. The most usual method was HTTP request filtering; DNS tampering (directing to a blackhole IP address) affected only the three domains facebook.com, youtube.com, and plus.google.com. SSH connections were throttled down to about 15% of the link capacity, while randomized protocols were throttled almost down to zero, 60 seconds into a connection's lifetime. Throttling seemed to be achieved by dropping packets, which causes TCP to slow down.

Khattak et al. [114] evaluated the Great Firewall from the perspective that it works like an intrusion detection system or network monitor, and applied existing techniques for evading a monitor to the problem of circumvention. They looked particularly for ways to evade detection that are expensive for the censor to remedy. They found that the firewall was stateful, but only in the client-to-server direction. The firewall was vulnerable to a variety of TCP- and HTTP-based evasion techniques, such as overlapping fragments, TTL-limited packets, and URL encodings.

Nabi [141] investigated web censorship in Pakistan in 2013, using a publicly available list of banned web sites. Over half of the sites on the list were blocked by DNS tampering; less than 2% were additionally blocked by HTTP filtering (an injected redirect before April 2013, or a static block page after that). They conducted a small survey to find the most commonly used circumvention methods; the most common was public VPNs, at 45% of respondents. Khattak et al. [115] looked at two censorship events that took place in Pakistan in 2011 and 2012. Their analysis is special because unlike most studies of censorship, theirs uses traffic

traces taken directly from an ISP. They observe that users quickly switched to TLS-based circumvention following a block of YouTube. The blocks had side effects beyond a loss of connectivity: the ISP had to deal with more ciphertext than before, and users turned to alternatives for the blocked sites. Their survey found that the most common method of circumvention was VPNs. Aceto and Pescapè [2] revisited Pakistan in 2016. Their analysis of six months of active measurements in five ISPs showed that blocking techniques differed across ISPs; some used DNS poisoning and others used HTTP filtering. They did their own survey of commonly used circumvention technologies, and again the winner was VPNs with 51% of respondents.

Ensafi et al. [61] employed an intriguing technique to measure censorship from many locations in China—a "hybrid idle scan." The hybrid idle scan allows one to test TCP connectivity between two Internet hosts, without needing to control either one. They selected roughly uniformly geographically distributed sites in China from which to measure connectivity to Tor relays, Tor directory authorities, and the web servers of popular Chinese web sites. There were frequent failures of the firewall resulting in temporary connectivity, typically occurring in bursts of hours.

In 2015, Marczak et al. [129] investigated an innovation in the capabilities of the border routers of China, an attack tool dubbed the Great Cannon. The cannon was responsible for denial-of-service attacks on Amazon CloudFront and GitHub. The unwitting participants in the attack were web browsers located *outside* of China, who began their attack when the cannon injected malicious JavaScript into certain HTTP responses originating inside of China. The new attack tool was noteworthy because it demonstrated previously unseen in-path behavior, such as packet dropping.

A major aspect of censor modeling is that many censors use commercial firewall hardware. Dalek et al. [39], Dalek et al. [38], and Marquis-Boire et al. [130] documented the use of commercial firewalls made by Blue Coat, McAfee, and Netsweeper in a number of countries. Chaabane et al. [27] analyzed 600 GB of leaked logs from Blue Coat proxies that were being used for censorship in Syria. The logs cover 9 days in July and August 2011, and contain an entry for every HTTP request. The authors of the study found evidence of IP address blocking, DNS blocking, and HTTP request keyword blocking; and also evidence of users circumventing censorship by downloading circumvention software or using cache feature of Google search. All subdomains of .il, the top-level domain for Israel, were blocked, as were many IP address ranges in Israel. Blocked URL keywords included "proxy", which resulted in collateral damage to the Google Toolbar and the Facebook like button because they included the string "proxy" in HTTP requests. Tor was only lightly censored: only one of several proxies blocked it, and only sporadically.

Generic measurement platforms

For a decade, the OpenNet Initiative produced reports on Internet filtering and surveillance in dozens of countries, until it ceased operation in 2014. For example, their 2005 report on Internet filtering in China [146] studied the problem from many perspectives, political, technical, and legal. They tested the extent of filtering of web sites, search engines, blogs, and email. They found a number of blocked web sites, some related to news and politics, and some on sensitive subjects such as Tibet and Taiwan. In some cases, entire domains

were blocked; in others, only specific URLs within the domain were blocked. There were cases of overblocking: apparently inadvertently blocked sites that happened to share an IP address or URL keyword with an intentionally blocked site. The firewall terminated connections by injecting a TCP RST packet, then injecting a zero-sized TCP window, which would prevent any communication with the same server for a short time. Using technical tricks, the authors inferred that Chinese search engines indexed blocked sites (perhaps having a special exemption from the general firewall policy), but did not return them in search results [147]. Censorship of blogs included keyword blocking by domestic blogging services, and blocking of external domains such as blogspot.com [145]. Email filtering was done by the email providers themselves, not by an independent network firewall. Email providers seemed to implement their filtering rules independently and inconsistently: messages were blocked by some providers and not others.

Sfakianakis et al. [169] built CensMon, a system for testing web censorship using PlanetLab, a distributed network research platform. They ran the system for 14 days in 2011 across 33 countries, testing about 5,000 unique URLs. They found 193 blocked domain–country pairs, 176 of them in China. CensMon was not run on a continuing basis. Verkamp and Gupta [185] did a separate study in 11 countries, using a combination of PlanetLab nodes and the computers of volunteers. Censorship techniques varied across countries; for example, some showed overt block pages and others did not.

OONI [92] and ICLab [107] are dedicated censorship measurement platforms. Razaghpanah et al. [159] provide a comparison of the two platforms. They work by running regular network measurements from the computers of volunteers or through VPNs. UBICA [3] is another system based on volunteers running probes; its authors used it to investigate several forms of censorship in Italy, Pakistan, and South Korea.

Anderson et al. [8] used RIPE Atlas a globally distributed Internet measurement network, to examine two case studies of censorship: Turkey's ban on social media sites in March 2014 and Russia's blocking of certain LiveJournal blogs in March 2014. Atlas allows 4 types of measurements: ping, traceroute, DNS resolution, and TLS certificate fetching. In Turkey, they found at least six shifts in policy during two weeks of site blocking. They observed an escalation in blocking in Turkey: the authorities first poisoned DNS for twitter.com, then blocked the IP addresses of the Google public DNS servers, then finally blocked Twitter's IP addresses directly. In Russia, they found ten unique bogus IP addresses used to poison DNS.

Pearce, Ensafi, et al. [150] made Augur, a scaled-up version of the hybrid idle scan of Ensafi et al. [61], designed for continuous, global measurement of disruptions of TCP connectivity. The basic tool is the ability to detect packet drops between two remote hosts; but expanding it to a global scale poses a number of technical challenges. Pearce et al.[151] built Iris, as system to measure DNS manipulation globally. Iris uses open resolvers and evaluates measurements against the detection metrics of consistency (answers from different locations should the same or similar) and independent verifiability (checking results against other sources of data like TLS certificates) in order to decide when they constitute manipulation.

3.2 The evaluation of circumvention systems

Evaluating the quality of circumvention systems is tricky, whether they are only proposed or actually deployed. The problem of evaluation is directly tied to threat modeling. Circumvention is judged according to how well it works under a given model; the evaluation is therefore meaningful only as far as the threat model reflects reality. Without grounding in reality, researchers risk running an imaginary arms race that evolves independently of the real one.

I took part, with Michael Carl Tschantz, Sadia Afroz, and Vern Paxson, in a meta-study [182] of how circumvention systems are evaluated by their authors and designers, and comparing those to empirically determined censor models. This kind of work is rather different than the direct evaluations of circumvention tools that have happened before, for example those done by the Berkman Center [162] and Freedom House [26] in 2011. Rather than testing tools against censors, we evaluated how closely aligned designers' own models were to models derived from actual observations of censors.

This research was partly born out of frustration with some typical assumptions made in academic research on circumvention, which we felt placed undue emphasis on steganography and obfuscation of traffic streams, while not paying enough attention to the perhaps more important problems of proxy distribution and initial rendezvous between client and proxy. We wanted to help bridge the gap by laying out a research agenda to align the incentives of researchers with those of circumventors. This work was built on extensive surveys of circumvention tools, measurement studies, and known censorship events against Tor. Our survey included over 50 circumvention tools.

One outcome of the research is that that academic designs tended to be concerned with detection in the steady state after a connection is established (related to detection by content), while actually deployed systems cared more about how the connection is established initially (related to detection by address). Designers tend to misperceive the censor's weighting of false positives and false negatives—assuming a whitelist rather than a blacklist, say. Real censors care greatly about the cost of running detection, and prefer cheap, passive, stateless solutions whenever possible. It is important to guard against these modes of detection before becoming too concerned with those that require sophisticated computation, packet flow blocking, or lots of state.

Chapter 4

Active probing

The Great Firewall of China rolled out an innovation in the identification of proxy servers around 2010: *active probing* of suspected proxy addresses. In active probing, the censor pretends to be a legitimate client, making its own connections to suspected addresses to see whether they speak a proxy protocol. Any addresses that are found to be proxies are added to a blacklist so that access to them will be blocked in the future. The input to the active probing subsystem, a set of suspected addresses, comes from passive observation of the content of client connections. The censor sees a client connect to a destination and tries to determine, by content inspection, what protocol is in use. When the censor's content classifier is unsure whether the protocol is a proxy protocol, it passes the destination address to the active probing subsystem. Active prober then checks, by connecting to the destination, whether it actually is a proxy. Figure 4.1 illustrates the process.

Active probing makes good sense for the censor, whose main restriction is the risk of false-positive classifications that result in collateral damage. Any classifier based purely on passive content inspection must be very precise (have a low rate of false positives). Active probing increases precision by blocking only those servers that are determined, through active inspection, to really be proxies. The censor can get away with a mediocre content classifier—it can return a superset of the set of actual proxy connections, and active probes will weed out its false positives. A further benefit of active probing, from the censor's point of view, is that it can run asynchronously, separate from the firewall's other responsibilities that require a low response time.

Active probing, as I use the term in this chapter, is distinguished from other types of active scans by being reactive, driven by observation of client connections. It is distinct from proactive, wide-scale port scanning, in which a censor regularly scans likely ports across the Internet to find proxies, independent of client activity. The potential for the latter kind of scanning has been appreciated for over a decade. Dingledine and Mathewson [49 §9.3] raised scanning resistance as a consideration in the design document for Tor bridges. McLachlan and Hopper [136 §3.2] observed that the bridges' tendency to run on a handful of popular ports would make them more discoverable in an Internet-wide scan, which they estimated would take weeks using then-current technology. Dingledine [46 §6] mentioned indiscriminate scanning as one of ten ways to discover Tor bridges—while also bringing up the possibility of reactive probing which the Great Firewall was then just beginning to use. Durumeric et al. [57 §4.4] demonstrated the effectiveness of Internet-wide scanning, targeting only two

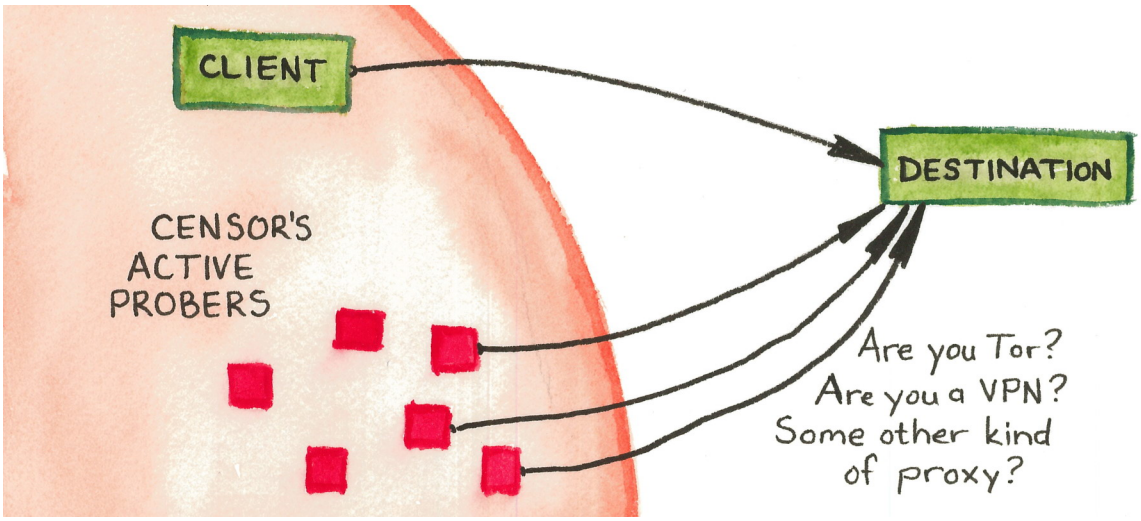

Figure 4.1: The censor watches a connection between a client and a destination. If content inspection does not definitively indicate the use of a circumvention protocol, but also does not definitively rule it out, the censor passes the destination's address to an active prober. The active prober attempts connections using various proxy protocols. If any of the proxy connections succeeds, the censor adds the destination to an address blacklist.

ports to discover about 80% of public Tor bridges in only a few hours, Tsyrklevich [183] and Matic et al. [133 §V.D] later showed how existing public repositories of Internet scan data could reveal bridges, without even the necessity of running one's own scan.

The Great Firewall of China is the only censor known to employ active probing. It has increased in sophistication over time, adding support for new protocols and reducing the delay between a client's connection and the sending of probes. The Great Firewall has the documented ability to probe the plain Tor protocol and some of its pluggable transports, as well as certain VPN protocols and certain HTTPS-based proxies. Probing takes place only seconds or minutes after a connection by a legitimate client, and the active-probing connections come from a large range of source IP addresses. The experimental results in this chapter all have to do with China.

Active probing occupies a space somewhere in the middle of the dichotomy, put forward in Chapter 2, of detection by content and detection by address. An active probing system takes suspected addresses as input and produces to-be-blocked addresses as output. But it is content-based classification that produces the list of suspected addresses in the first place. The existence of active probing is The use of active probing is, in a sense, a good sign for circumvention: it only became relevant content obfuscation had gotten better. If a censor could easily identify the use of circumvention protocols by mere passive inspection, then it would not go to the extra trouble of active probing.

Contemporary circumvention systems must be designed to resist active probing attacks. The strategy of the look-like-nothing systems ScrambleSuit [200], obfs4 [206], and Shadowsocks [126, 156] is to authenticate clients using a per-proxy password or public key; i.e., to require some additional secret information beyond just an IP address and port number. Domain fronting (Chapter 6) deals with active probing by co-locating proxies with important

2010 August	Nixon notices unusual, random-looking connections from China in SSH logs [143].
2011 May–June	Nixon's random-looking probes are temporarily replaced by TLS probes before changing back again [143].
2011 October	hrimfaxi reports that Tor bridges are quickly detected by the GFW [41].
2011 November	Nixon publishes observations and hypotheses about the strange SSH connections [143].
2011 December	Wilde investigates Tor probing [48, 193, 194]. He finds two kinds of probe: "garbage" random probes and Tor-specific ones.
2012 February	The obfs2 transport becomes available [43]. The Great Firewall is initially unable to probe for it.
2012 March	Winter and Lindskog investigate Tor probing in detail [199].
2013 January	The Great Firewall begins to active-probe obfs2 [47, 60 §4.3]. The obfs3 transport becomes available [68].
2013 June–July	Majkowski observes TLS and garbage probes and identifies fingerprintable features of the probers [128].
2013 August	The Great Firewall begins to active-probe obfs3 [60 Figure 8].
2014 August	The ScrambleSuit transport, which is resistant to active probing, becomes available [153].
2015 April	The obfs4 transport (resistant to active probing) becomes available [154].
2015 August	BreakWa11 finds an active-probing vulnerability in Shadowsocks [19, 156 §2].
2015 October	Ensafi et al. [60] publish results of multi-modal experiments on active probing.
2017 February	Shadowsocks changes its protocol to better resist active probing [102].
2017 May	Wang et al. [189 §7.3] find that bridges that are discovered by active probing are blocked on the entire IP address, not an individual port.

Table 4.2: Timeline of research on active probing.

web services: the censor can tell that circumvention is taking place but cannot block the proxy without unacceptable collateral damage. In Snowflake (Chapter 7), proxies are web browsers running ordinary peer-to-peer protocols, authenticated using a per-connection shared secret. Even if a censor discovers one of Snowflake's proxies, it cannot verify that the proxy is running Snowflake or something else, without having first negotiated a shared secret through Snowflake's broker mechanism.

4.1 History of active probing research

Active probing research has mainly focused on Tor and its pluggable transports. There is also some work on Shadowsocks. Table 4.2 summarizes the research covered in this section.

Nixon [143] published in late 2011 an analysis of suspicious connections from IP addresses in China that his servers had at that point been receiving for a year. The connections were to the SSH port, but did not follow the SSH protocol; rather they contained apparently random

bytes, resulting in error messages in the log file. Nixon discovered a pattern: the random-looking "garbage" probes were preceded, at an interval of 5–20 seconds, by a legitimate SSH login from some other IP address in China. The same pattern was repeated at three other sites. Nixon supposed that the probes were triggered by legitimate SSH users, as their connections traversed the firewall; and that the random payloads were a simple form of service identification, sent only to see how the server would respond to them. For a few weeks in May and June 2011, the probes did not look random, but instead looked like TLS.

In October 2011, Tor user hrimfaxi reported that a newly set up, unpublished Tor bridge would be blocked within 10 minutes of their first being accessed from China [41]. Moving the bridge to another port on the same IP address would work temporarily, but the new address would also be blocked within another 10 minutes. Wilde systematically investigated the phenomenon in December 2011 and published an extensive analysis of active probing that was triggered by connections from inside China to outside [193, 194]. There were two kinds of probes: "garbage" random probes like those Nixon had described, and specialized Tor probes that established a TLS session and inside the session sent the Tor protocol. The garbage probes were triggered by TLS connections to port 443, and were sent immediately following the original connection. The Tor probes, in contrast, were triggered by TLS connections to any port, as long as the TLS client handshake matched that of Tor's implementation [48]. The Tor probes were not sent immediately, but in batches of 15 minutes. The probes came from diverse IP addresses in China: 20 different /8 networks [192]. Bridges using the obfs2 transport were, at that time, neither probed nor blocked.

Winter and Lindskog revisited the question of Tor probing a few months later in 2012 [199]. They used open proxies and a server in China to reach bridges and relays in Russia, Singapore, and Sweden. The bridges and relays were configured so that ordinary users would not connect to them by accident. They confirmed Wilde's finding that the blocking of one port did not affect other ports on the same IP address. Blocked ports became reachable again 12 hours. By simulating multiple Tor connections, they collected over 3,000 active probe samples in 17 days. During that time, there were about 72 hours which where mysteriously free of active probing. Half of the probes came from a single IP address, 202.108.181.70; the other half were almost all unique. Reverse-scanning the probes' source IP addresses, a few minutes after the probes were received, sometimes found a live host, though usually with a different IP TTL than was used during the probing, which the authors suggested may be a sign of address spoofing by the probing infrastructure. Because probing was triggered by patterns in the TLS client handshake, they developed a server-side tool, brdgrd [196], that rewrote the TCP window so that the client's handshake would be split across packets. The tool sufficed, at the time, to prevent active probing, but stopped working in 2013 [197 §Software].

The obfs2 pluggable transport, first available in February 2012 [43], worked against active probing for about a year. The first report of its being probing arrived in March 2013 [47]. I found evidence for an even earlier onset, in January 2013, by analyzing the logs of my web server [60 §4.3]. At about the same time, the obfs3 pluggable transport became available [68]. It was, in principle, as vulnerable to active probing as obfs2 was, but the firewall did not gain the ability to probe for it until August 2013 [60 Figure 8].

Majkowski [128] documented a change in the GFW between June and July 2013. In June, he reproduced the observations of Winter and Lindskog: pairs of TLS probes, one from 202.108.181.70 and one from some other IP address. He also provided TLS fingerprints for

the probers, which differed from those of ordinary Tor clients. In July, he began to see pairs of probes with apparently random contents, like the garbage probes Wilde described. The TLS fingerprints of the July probes differed from those seen earlier, but were still distinctive.

The ScrambleSuit transport, designed to be immune to active-probing attacks, first shipped with Tor Browser 4.0 in October 2014 [153]. The successor transport obfs4, similarly immune, shipped in Tor Browser 4.5 in April 2015 [154].

In August 2015, developer BreakWa11 described an active-probing vulnerability in the Shadowsocks protocol [19, 156 §2]. The flaw had to do with a lack of integrity protection, allowing a prober to introduce errors into ciphertext and watch the server's reaction. As a stopgap, the developers deployed a protocol change that proved to have its own vulnerabilities to probing. They deployed another protocol in February 2017, adding cryptographic integrity protection and fixing the problem [102]. Despite the long window of vulnerability, I know of no evidence that the Great Firewall tried to probe for Shadowsocks servers.

Ensafi et al. (including me) [60] did the largest controlled study of active probing to date throughout early 2015. We collected data from a variety of sources: a private network of our own bridges, isolated so that only we and active probers would connect to them; induced intensive probing of a single bridge over a short time period, in the manner of Winter and Lindskog; analysis of server log files going back to 2010; and reverse-scanning active prober source IP addresses using tools such as ping, traceroute, and Nmap. Using these sources of data, we investigated many aspects of active probing, such as the types of probes the firewall was capable of sending, the probers' source addresses, and potentially fingerprintable peculiarities of the probers' protocol implementations. Observations from this research project appear in the remaining sections of this chapter.

Wang et al. [189 §7.3] tried connecting to bridges from 11 networks in China. They found that connections from four of the networks did not result in active probing, while connections from the other seven did. A bridge that was probed became blocked on all ports, a change from the single-port blocking that had been documented earlier.

4.2 Types of probes

Our experiments confirmed the existence of known probe types from prior research, and new types that had not been documented before. Our observations of the known probe types were consistent with previous reports, with only minor differences in some details. We found, at varying times, these kinds of probes:

Tor We found probing of the Tor protocol, as expected. The probes we observed in 2015, however, differed from those Wilde described in 2011, which proceeded as far as building a circuit. The ones we saw used less of the Tor protocol: after the TLS handshake they only queried the server's version and disconnected. Also, in contrast to what Winter and Lindskog found in 2012, the probes were sent immediately after a connection, not batched to a multiple of 15 minutes.

obfs2 The obfs2 protocol is meant to look like a random stream, but it has a weakness that makes it trivial to identify, passively and retroactively, needing only the first 20 bytes sent by the client. We turned the weakness of obfs2 to our advantage. It allowed us to

distinguish obfs2 from other random-looking payloads, isolating a set of connections that could belong only to legitimate circumventors or to active probers.

obfs3 The obfs3 protocol is also meant to look like a random stream; but unlike obfs2, it is not trivially identifiable passively. It is not possible to retroactively recognize obfs3 connections (from, say, a packet capture) with certainty: sure classification requires active participation in the protocol. In some of our experiments, we ran an obfs3 server that was able to participate in the handshake and so confirm that the protocol really was obfs3. In the passive log analysis, we labeled "obfs3" any probes that looked random but were not obfs2.

SoftEther We unexpectedly found evidence of probe types other than Tor-related ones. One of these was an HTTPS request:

```
POST /vpnsvc/connect.cgi HTTP/1.1
Connection: Keep-Alive
Content-Length: 1972
Content-Type: image/jpeg

GIF89a...
```

Both the path "/vpnsvc/connect.cgi", and the body being a GIF image despite having a Content-Type of "image/jpeg", are characteristic of the client handshake of the SoftEther VPN software that underlies the VPN Gate circumvention system [144].

AppSpot This type of probe is also an HTTPS request:

```
GET / HTTP/1.1
Accept-Encoding: identity
Connection: close
Host: webncsproxyXX.appspot.com
Accept: */*
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
  Gecko) Ubuntu Chromium/34.0.1847.116 Chrome/34.0.1847.116 Safari/537.36
```

where the '`XX`' is a number that varies. The intent of this probe seems to be the discovery of servers that are capable of domain fronting for Google services, including Google App Engine, which runs at appspot.com. (See Chapter 6 for more on domain fronting.) At one time, there were simple proxies running at webncsproxyXX.appspot.com.

urllib This probe type is new since our 2015 paper. I discovered it while re-analyzing my server logs in order to update Figure 4.3. It is a particular request that was sent over both HTTP and HTTPS:

```
GET / HTTP/1.1
Accept-Encoding: identity
Host: 69.164.193.231
Connection: close
User-Agent: Python-urllib/2.7
```

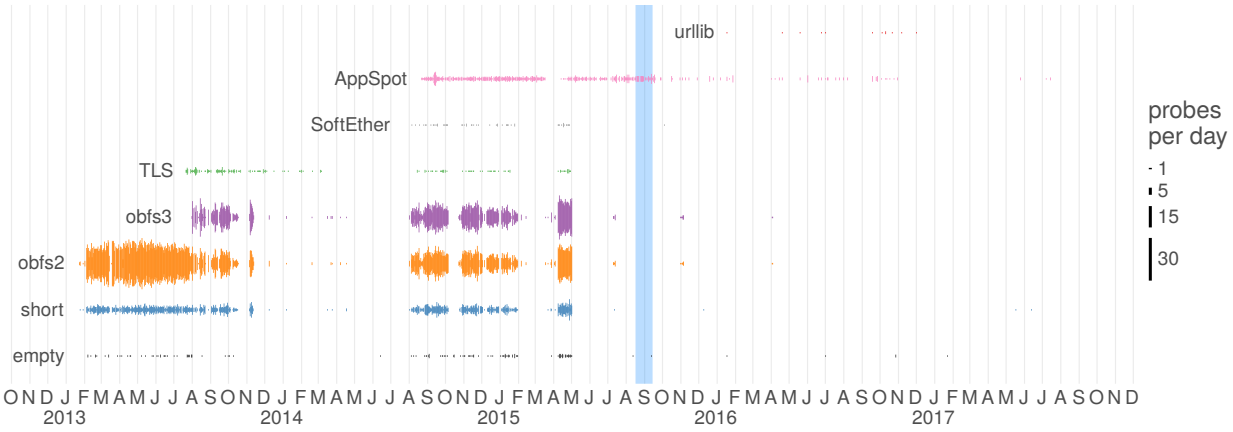

Figure 4.3: Active probes received at my web server (ports 80 and 443) over five years. This is an updated version of Figure 8 from the paper "Examining How the Great Firewall Discovers Hidden Circumvention Servers" [60]; the vertical blue stripe divides old and new data. The "short" probes are those that looked random but did not provide enough data (20 bytes) for the obfs2 test; it is likely that they, along with the "empty" probes, are really truncated obfs2, obfs3, or Tor probes. The traffic from the IP addresses represented in this chart was overwhelmingly composed of active probes, but there were also 97 of what looked like genuine client browser requests. Active probing activity—at least against this server—has subsided since 2016.

> The urllib requests are unremarkable except for having been sent from an IP address that at some other time send another kind of active probe. The User-Agent "Python-urllib/2.7" and appears many other places in my logs, not in an active probing context. I cannot guess what this probe's purpose may be, except to observe that Nobori and Shinjo also caught a "Python-urllib" client scraping the VPN Gate server list [144 §6.3].

These probe types are not necessarily exhaustive. The purpose of the random "garbage" probes is still not known; they were not obfs2 and were too early to be obfs3, so they must have had some other purpose.

Most of our experiments were designed around exploring known Tor-related probe types: plain Tor, obfs2, and obfs3. The server log analysis, however, unexpectedly turned up the other probe types. The server log data set consisted of application-layer logs from my personal web and mail server, which was also a Tor bridge. Application-layer logs lack much of the fidelity you would normally want in a measurement experiment; they do not have precise timestamps or transport-layer headers, for example, and web server logs truncate the client's payload at the first '\0' or '\n' byte. But they make up for that with time coverage. Figure 4.3 shows the history of probes received at my server since 2013 (there were no probes before that, though the logs go back to 2010). We began by searching the logs for definite probes: those that were classifiable as obfs2 or otherwise looked like random garbage. Then we looked for what else appeared in the logs for the IP addresses that had, at any time, sent a probe. In a small fraction of cases, the other logs lines appeared to be genuine HTTP requests from legitimate clients; but usually they were other probe-like payloads. We continued this process, adding new classifiers for likely probes, until reaching a fixed point with the probe types described above.

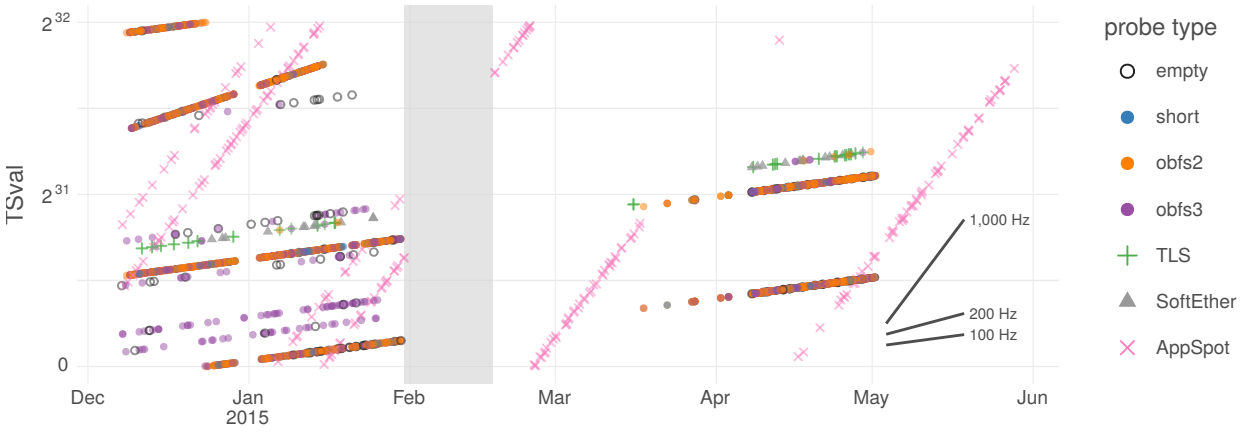

Figure 4.4: TCP timestamp values of active probes. A TCP timestamp is a 32-bit counter that increases at a constant rate [18 §5.4]; a sequence of timestamps is characterized by its rate and starting offset. There are 4,239 probes from 3,797 different source IP addresses depicted in the graph; however there are only a few distinct TCP timestamp sequences. There are three rates of increase (different slopes): 100 Hz, 200 Hz, and 1,000 Hz. The shaded area marks a gap in packet capture.

4.3 Probing infrastructure

The most striking feature of active probes is the large number of source addresses from which they are sent, or appear to be sent. The 13,089 probes received by the HTTP and HTTPS ports of my server came from 11,907 distinct IP addresses, representing 47 /8 networks and 26 autonomous systems. 96% of the addresses appeared only once. There is one extreme outlier, the address 202.108.181.70, which by itself accounted for 2% of the probes. (Even this substantial fraction stands in contrast to previous studies, where that single IP address accounted for roughly half the probes [199 §4.5.1].) Among the address ranges are ones belonging to residential ISPs.

Despite the many source addresses, the probes seems to be managed by only a few underlying processes. The evidence for this lies in shared patterns in metadata: TCP initial sequence numbers and TCP timestamps. Figure 4.4 shows clear patterns in TCP timestamps, from about six months during which we ran a full packet capture on my web server, in addition to application-layer logging.

We tried connecting back to the source address of probes. Immediately after receiving a probe, the probing IP address would be completely unresponsive to any stimulus we could think to apply. In some cases though, within an hour the address became responsive. The responsive hosts looked like what you would expect to find if you scanned such address ranges, with a variety of operating systems and open ports.

4.4 Fingerprinting the probers

A potential countermeasure against active probing is for each proxy, when it receives a connection, to somehow decide whether the connection comes from a legitimate client, or from a prober. Of course, the right way to identify legitimate clients is with cryptographic

authentication, whether at the transport layer (like BridgeSPA [172]) or at the application layer (like ScrambleSuit, obfs4, and Shadowsocks). But when that is not possible, one might hope to distinguish probers by their fingerprints, idiosyncrasies in their implementation that make them stand out from ordinary clients. In the case of the Great Firewall, the source IP address does not suffice as a fingerprint because of the great diversity of source addresses the system makes use of. And in a reversal of the usual collateral damage, the source addresses include those where we might expect legitimate clients to reside. The probes do, however, exhibit certain fingerprints at the application layer. While none of the ones we found were robust enough to effectively exclude active probers, they do hint at how the probing is implemented.

The active probers have an unusual TLS fingerprint, TLSv1.0 with a peculiar list of ciphersuites. Tor probes sent only a VERSIONS cell [50 §4.1], waited for a response, then closed the connection. The format of the VERSIONS cell was that of a "v2" Tor handshake that has been superseded since 2011, though still in use by a small number of real clients. The Tor probes described by Wilde in 2011 went further into the protocol. It hints at the possibility that at one time, the active probers used a (possibly modified) Tor client, and later switched to a custom implementation.

The obfs2 probes were conformant with the protocol specification, and unremarkable except for the fact that sometimes payloads were duplicated. obfs2 clients are supposed to use fresh randomness for each connection, but a small fraction, about 0.65%, of obfs2 probes shared an identical payload with one other probe. The two probes in a pair came from different source IP addresses and arrived within a second of each other. The apparently separate probers must therefore share some state—at least a shared pseudorandom number generator.

The obfs3 protocol calls for the client to send a random amount of random bytes as padding. The active probers' implementation of the protocol gets the probability distribution wrong, half the time sending too much padding. This feature would be difficult to exploit for detection, though, because it would rely on application-layer proxy code being able to infer TCP segment boundaries.

The SoftEther probes seem to have been based on an earlier version of the official SoftEther client software than was current at the time, differing from current version in that they lack an HTTP Host header. They also differed from the official client in that their POST request was not preceded by a GET request. The TLS fingerprint of the official client is much different from that of the probers, again hinting at a custom implementation.

The AppSpot probes have a User-Agent header that claims to be a specific version of the Chrome browser; however the rest of the header, and the TLS fingerprint are inconsistent with Chrome.

Chapter 5

Time delays in censors' reactions

Censors' inner workings are mysterious. To the researcher hoping to understand them they present only a hostile, black-box interface. However some of their externally visible behaviors offers hints about their internal decision making. In this chapter I describe the results of an experiment that is designed to shed light on the actions of censors; namely, a test of how quickly they react to and block a certain kind of Tor bridge.

Tor bridges are secret proxies that help clients get around censorship. The effectiveness of bridges depends on their secrecy—a censor that learns a bridge's address can simply block its IP address. Since the beginning, the designers of Tor's bridge system envisioned that users would learn of bridges through covert or social channels [49 §7], in order to prevent any one actor from learning about and blocking a large number of them.

But as it turns out, most users do not use bridges in the way envisioned. Rather, most users who use bridges use one of a small number of *default* bridges hardcoded in a configuration file within Tor Browser. (According to Matic et al. [133 §VII.C], over 90% of bridge users use a default bridge.) At a conceptual level, the notion of a "default" bridge is a contradiction: bridges are meant to be secret, not plainly listed in the source code. Any reasonable threat model would assume that default bridges are immediately blocked. And yet in practice we find that they are often not blocked, even by censors that otherwise block Tor relays. We face a paradox: why is it that censors do not take blocking steps that we find obvious? There must be some quality of censors' internal dynamics that we do not understand adequately.

The purpose of this chapter is to begin to go beneath the surface of censorship for insight into why censors behave as they do—particularly when they behave contrary to expectations. We posit that censors, far from being unitary entities of focused purpose, are rather complex organizations composed of human and machine components, with perhaps conflicting goals; this project is a small step towards better understanding what lies under the face that censors present. The main vehicle for the exploration of this subject is the observation of default Tor bridges to find out how quickly they are blocked after they first become discoverable by a censor. I took part in this project along with Lynn Tsai and Qi Zhong; the results in this chapter are an extension of work Lynn and I published in 2016 [91]. Through active measurements of default bridges from probe sites in China, Iran, and Kazakhstan, we uncovered previously undocumented behaviors of censors that hint at how they operate at a deeper level.

It was with a similar spirit that Aase, Crandall, Díaz, Knockel, Ocaña Molinero, Saia,

Wallach, and Zhu [1] looked into case studies of censorship with a focus on understanding censors' motivation, resources, and time sensitivity. They "had assumed that censors are fully motivated to block content and the censored are fully motivated to disseminate it," but some of their observations challenged that assumption, with varied and seemingly undirected censorship hinting at behind-the-scenes resource limitations. They describe an apparent "intern effect," by which keyword lists seem to have been compiled by a bored and unmotivated worker, without much guidance. Knockel et al. [117] looked into censorship of keywords in Chinese mobile games, finding that censorship enforcement in that context is similarly decentralized, different from the centralized control we commonly envision when thinking about censorship.

Zhu et al. [207] studied the question of censor reaction time in a different context: deletion of posts on the Chinese microblogging service Sina Weibo. Through frequent polling, they were able to measure—down to the minute—the delay between when a user made a post and when a censor deleted it. About 90% of deletions happened within 24 hours, and 30% within 30 minutes—but there was a long tail of posts that survived several weeks before being deleted. The authors used their observations to make educated guesses about the inner workings of the censors. Posts on trending topics tended to be deleted more quickly. Posts made late at night had a longer average lifetime, seemingly reflecting workers arriving in the morning and clearing out a nightly backlog of posts. King et al. [116] examined six months' worth of deleted posts on Chinese social networks. The pattern of deletions seemed to reveal the censor's motivation: not to prevent criticism of the government, as might be expected, but to forestall collective public action.

Nobori and Shinjo give a timeline [144 §6.3] of circumventor and censor actions and reactions during the first month and a half of the deployment of VPN Gate in China. Within the first four days, the firewall had blocked their main proxy distribution server, and begun scraping the proxy list. When they blocked the single scraping server, the firewall began scraping from multiple other locations within a day. After VPN Gate deployed the countermeasure of mixing high-collateral-damage servers into their proxy list, the firewall stopped blocking for two days, then resumed again, with an additional check that an IP addresses really was a VPN Gate proxy before blocking it.

Wright et al. [202 §2] motivated a desire for fine-grained censorship measurement by highlighting limitations that tend to prevent a censor from begin equally effective everywhere in its controlled network. Not only resource limitations, but also administrative and logistical requirements, make it difficult to manage a system as complex as a national censorship apparatus.

There has been no prior long-term study dedicated to measuring time delays in the blocking of default bridges. There have, however, been a couple of point measurements that put bounds on what blocking delays in the past must have been. Tor Browser first shipped with default obfs2 bridges on February 11, 2012 [43]; Winter and Lindskog tested them 41 days later [199 §5.1] and found all 13 of them blocked. (The bridges then were blocked by RST injection, a different blocking technique than the timeouts we have seen more recently.) In 2015 I used public reports of blocking and non-blocking of the first batch of default obfs4 bridges to infer a blocking delay of not less than 15 and not more than 76 days [70].

As security researchers, are accustomed to making conservative assumptions when building threat models. For example, we assume that when a computer is compromised, it's game

over: the attacker will cause the worst possible outcome for the computer's owner. But the actual effects of a compromise can vary from grave to almost benign, and it is an interesting question, what really happens and how severe it is. Similarly, it is prudent to assume while modeling that the disclosure of any secret bridge will result in its immediate blocking by every censor everywhere. But as that does not happen in practice, it is an interesting question, what really does happen, and why?

5.1 The experiment

Our experiment primarily involved frequent, active test of the reachability of default bridges from probe sites in China, Iran, and Kazakhstan (countries well known to censor the network), as well as a control site in the U.S. We used a script that, every 20 minutes, attempted to make a TCP connection to each default bridge. The script recorded, for each attempt, whether the connection was successful, the time elapsed, and any error code. The error code allows us to distinguish between different kinds of failures such as "timeout" and "connection refused." The control site in the U.S. enables us to distinguish temporary bridge failures from actual blocking.

The script only tested whether it is possible to make a TCP connection, which is a necessary but not sufficient precondition to actually establishing a Tor circuit through the bridge. In Kazakhstan, we deployed an additional script that attempted to establish a full Tor-in-obfs4 connection, in order to better understand the different type of blocking we discovered there.

The experiment was opportunistic in nature: we ran from China, Iran, and Kazakhstan not only because they are likely suspects for Tor blocking, but because we happened to have access to a site in each from which we could run probes over some period of time. Therefore the measurements cover different dates in different countries. We began at a time when Tor was building up its stock of default bridges. We began monitoring each new bridges as it was added, coordinating with the Tor Browser developers to get advance notice of their addition when possible. Additionally we had the developers run certain more controlled experiments for us—such as adding a bridge to the source code but commenting it out—that are further detailed below.

We were only concerned with default bridges, not secret ones. Our goal was not to estimate the difficulty of the proxy discovery problem, but to better understand how censors deal with what should be an easy task. We focused on bridges using the obfs4 pluggable transport [206], which not only is the most-used transport and the one marked "recommended" in the interface, but also has properties that help in our experiment. The content obfuscation of obfs4 reduces the risk of its passive detection. More importantly, it resists active probing attacks as described in Chapter 4. We could not have done the experiment with obfs3 bridges, because whether default or not, active probing would cause them to be blocked shortly after their first use.

Bridges are identified by a nickname and a port number. The nickname is an arbitrary identifier, chosen by the bridge operator. So, for example, "ndnop3:24215" is one bridge, and "ndnop3:10527" is another on the same IP address. We pulled the list of bridges from Tor Browser and Orbot, which is the port of Tor for Android. Tor Browser and Orbot mostly

New Tor Browser default obfs4 bridges

ndnop3	24215, 10527
ndnop5	13764
riemann	443
noether	443
Mosaddegh	41835, 80, 443, 2934, 9332, 15937
MaBishomarim	49868, 80, 443, 2413, 7920, 16488
GreenBelt	60873, 80, 443, 5881, 7013, 12166
JonbesheSabz	80, 1894, 4148, 4304
Azadi	443, 4319, 6041, 16815
Lisbeth	443
NX01	443
LeifEricson	50000, 50001, 50002
cymrubridge31	80
cymrubridge33	80

Orbot-only default obfs4 bridges

Mosaddegh	1984
MaBishomarim	1984
JonbesheSabz	1984
Azadi	1984

Already existing default bridges

LeifEricson	41213	(obfs4)
fdctorbridge01	80	(FTE)

Never-published bridges

ndnop4	27668	(obfs4)

Table 5.1: The bridges whose reachability we tested. Except for the already existing and never-published bridges, they were all introduced during the course of our experiment. We also tested port 22 (SSH) on hosts that had it open. Each bridge is identified by a nickname (a label chosen by its operator) and a port. Each nickname represents a distinct IP address. Port numbers are in chronological order of release.

shared bridges in common, though there were a few Orbot-only bridges. A list of the bridges and other destinations we measured appears in Table 5.1. Along with the fresh bridges, we tested some existing bridges for comparison purposes.

There are four stages in the process of deploying a new default bridge. At the beginning, the bridge is secret, perhaps having been discussed on a private mailing list. Each successive stage of deployment makes the bridge more public, increasing the number of places where a censor may look to discover it. The whole process takes a few days to a few weeks, mostly depending on Tor Browser's release schedule.

Ticket filed The process begins with the filing of a ticket in Tor's public issue tracker. The ticket includes the bridge's IP address. A censor that pays attention to the issue tracker could discover bridges as early as this stage.

Ticket merged After review, the ticket is merged and the new bridge is added to the source code of Tor Browser. From there it will begin to be included in nightly builds. A censor that reads the bridge configuration file from the source code repository, or downloads nightly builds, could discover bridges at this stage.

Testing release Just prior to a public release, Tor Browser developers send candidate builds to a public mailing list to solicit quality assurance testing. A censor that follows testing releases would find ready-made executables with embedded bridges at this stage. Occasionally the developers skip the testing period, such as in the case of an urgent security release.

Public release After testing, the releases are made public and announced on the Tor Blog. A censor could learn of bridges at this stage by reading the blog and downloading

executables. This is also the stage at which the new bridges begin to have an appreciable number of users. There are two release tracks of Tor Browser: stable and alpha. Alpha releases are distinguished by an 'a' in their version number, for example 6.5a4. According to Tor Metrics [180], stable downloads outnumber alpha downloads by a factor of about 30 to 1.

We advised operators to configure their bridges so that they would not become public except via the four stages described above. Specifically, we made sure the bridges did not appear in BridgeDB [181], the online database of secret bridges, and that the bridges did not expose any transports other than obfs4. We wanted to ensure that any blocking of bridges could only be the result of their status as default bridges, and not a side effect of some other detection system.

5.2 Results from China

We had access to probe sites in China for just over a year, from December 2015 to January 2017. Due to the difficulty of getting access to hosts in China, we used four different IP addresses (all in the same autonomous system) at different points in time. The times during which we had control of each IP address partially overlap, but there is a 21-day gap in the measurements during August 2016.

Our observations in China turned up several interesting behaviors of the censor. Throughout this section, refer to Figure 5.2, which shows the timeline of reachability of every bridge, in context with dates related to tickets and releases. Circled references in the text (ⓐ, ⓑ, etc.) refer to marked points in the figure. A "batch" is a set of Tor Browser releases that all contained the same default bridges.

The most significant single event—covered in detail in subsection 5.2.7—was a change in the censor's detection and blocking strategy in October 2016. Before that date, blocking was port-specific and happened only after the "public release" stage. After, bridges began to be blocked on all ports simultaneously, and were blocked soon after the "ticket merged" stage. We believe that this change reflects a shift in how the censor discovered bridges, a shift from running the finished software to see what addresses it accesses, to extracting the addresses from source code. More details and evidence appear in the following subsections.

5.2.1 Per-port blocking

In the first few release batches, the censor blocked individual ports, not an entire IP address. For example, see point ⓐ in Figure 5.2: after ndnop3:24215 was blocked, we opened ndnop3:10527 on the same IP address. The alternate port remained reachable until it, too, was blocked in the next release batch. We used this technique of rotating ports in several release batches.

Per-port blocking is also evident in the continued reachability of non-bridge ports. For example, many of the bridges had an SSH port open, in addition to their obfs4 ports. After riemann:443 (obfs4) was blocked (point ⓒ in Figure 5.2), riemann:22 (SSH) remained reachable for a further nine months, until it was finally blocked at point ⓜ. Per-port blocking would give way to whole-IP blocking in October 2016.

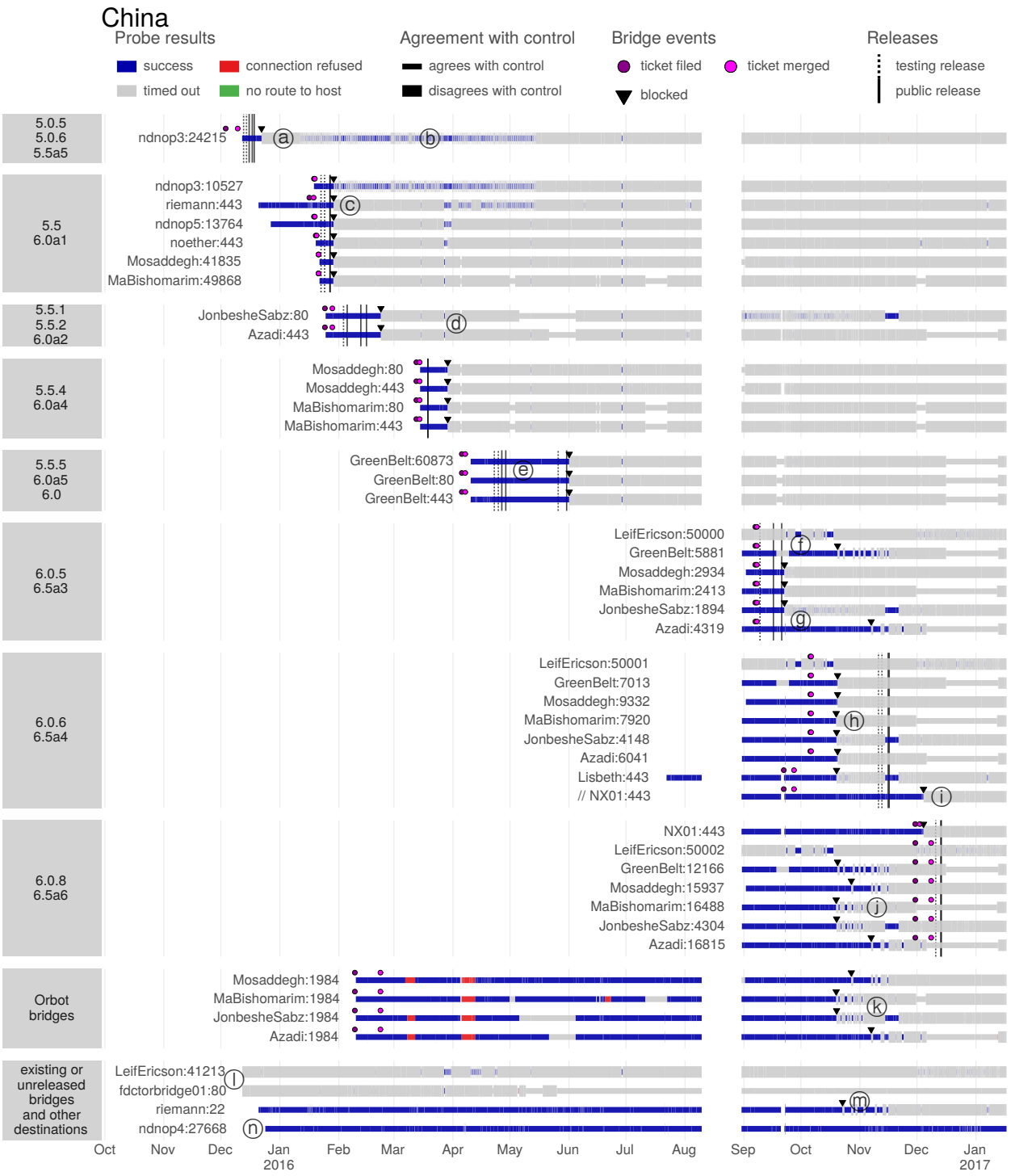

Figure 5.2: Default bridge reachability from a site in China. Releases are grouped into batches according to the new bridges they contain. The thickness of lines indicates whether the measurements agreed with those of the control site; their color shows whether the attempted TCP connection was successful. Blocking events appear as a transition from narrow blue (success, agrees with control) to wide gray (timeout, disagrees with control). The notation "// NX01:443" means that the bridge was commented out for that release. Points marked with circled letters ⓐ, ⓑ, etc., are explained in the text.

5.2.2 Blocking only after public release

In the first six batches, blocking occurred only after public release—despite the fact that the censor could potentially have learned about and blocked the bridges in an earlier stage. In the 5.5.5/6.0a5/6.0 batch, the censor even seems to have missed the 5.5.5 and 6.0a5 releases (point ⓔ in Figure 5.2), only blocking after the 6.0 release, 36 days later. This observation hints that, before October 2016 anyway, the censor was somehow extracting bridge addresses from the release packages themselves. In subsections 5.2.3 and 5.2.6 we present more evidence that supports the hypothesis that the censor extracted bridge addresses only from public releases, not reacting at any earlier phase.

An evident change in blocking technique occurred around October 2016 with the 6.0.6/6.5a4 batch, when for the first time bridge were blocked before a public or testing release was available. The changed technique is the subject of subsection 5.2.7.

5.2.3 Simultaneous blocking of all bridges in a batch

The first five blocking incidents were single events: when a batch contained more than one bridge, all were blocked at the same time; that is, within one of our 20-minute probing periods. These incidents appear as crisp vertical columns of blocking icons in Figure 5.2, for example at point ⓒ. This fact supports the idea that the censor discovered bridges by examining released executables directly, and did not, for example, detect bridges one by one by examining network traffic.

The 6.0.5/6.5a3 batch is an exception to the pattern of simultaneous blocking. In that batch, one bridge (LeifEricson:50000) was already blocked, three were blocked simultaneously as in the previous batches, but two others (GreenBelt:5881 and Azadi:4319) were temporarily unscathed. At the time, GreenBelt:5881 was experiencing a temporary outage—which could explain why it was not blocked—but Azadi:4319 was operational. This specific case is discussed further in subsection 5.2.6.

5.2.4 Variable delays before blocking

During the time when the censor was blocking bridges simultaneously after a public release, we found no pattern in the length of time between the release and the blocking event. The blocking events did not seem to occur after a fixed length of time, nor did they occur on the same day of the week or at the same time of day. The delays were 7, 2, 18, 10, 35, and 6 days after a batch's first public release—up to 57 days after the filing of the first ticket. Recall from Section 4.3 that the firewall was even at that time capable of detecting and blocking *secret* bridges within minutes. Delays of days or weeks stand out in contrast.

5.2.5 Inconsistent blocking and failures of blocking

There is a conspicuous on–off pattern in the reachability of certain bridges from China, for example in ndnop3:24215 throughout February, March, and April 2016 (point ⓑ in Figure 5.2). Although the censor no doubt intended to block the bridge fully, 47% of connection attempts were successful during that time. On closer inspection, we find that the pattern is roughly

periodic with a period of 24 hours. The pattern may come and go, for example in riemann:443 before and after March 27, 2016. The predictable daily variation in reachability rates makes us think that, at least at the times under question, the Great Firewall's effectiveness was dependent on load—varying load at different times of day leads to varying bridge reachability.

Beyond the temporary reachability of individual bridges, we also see what are apparent temporary failures of firewall, making all bridges reachable for hours or days at a time. Point ⓓ in Figure 5.2 marks such a failure. All the bridges under test, including those that had already been blocked, became available between 10:00 and 18:00 UTC on March 27, 2016. Further evidence that these results indicate a failure of the firewall come from a press report [101] that Google services—normally blocked in China—were also unexpectedly available on the same day, from about 15:30 to 17:15 UTC. A similar pattern appears across all bridges for nine hours starting on June 28, 2016 at 17:40 UTC.

After the switch to whole-IP blocking, there are further instances of spotty and inconsistent censorship, though of a different nature. Several cases are visible near point ⓙ in Figure 5.2. It is noteworthy that not all ports on a single host are affected equally. For example, the blocking of GreenBelt is inconsistent on ports 5881 and 12166, but it is solidly blocked on ports 80, 443, 7013, and 60873. Similarly, Mosaddegh's ports 1984 and 15937 are intermittently reachable, in the exact same pattern, while ports 80, 443, 2934, and 9332 remain blocked. These observations lead us to suspect a model of two-tiered blocking: one tier for per-port blocking, and a separate tier for whole-IP blocking. If there were a temporary failure of the whole-IP tier, any port not specifically handled by the per-port tier would become reachable.

5.2.6 Failure to block all new bridges in a batch

The 6.0.5/6.5a2 release batch was noteworthy in several ways. Its six new bridges were all fresh ports on already-used IP addresses. For the first time, not all bridges were blocked simultaneously. Only three of the bridges—Mosaddegh:2934, MaBishomarim:2413, and JonbesheSabz:1894—were blocked in a way consistent with previous release batches. Of the other three:

- LeifEricson:50000 had been blocked since we began measuring it. The LeifEricson IP address is one of the oldest in the browser. We suspect the entire IP address had been blocked at some point. We will have more to say about LeifEricson in subsection 5.2.8.

- GreenBelt:5881 (point ⓕ) was offline at the time when other bridges in the batch were blocked. We confirmed this fact by talking with the bridge operator and through control measurements: the narrow band in Figure 5.2 shows that connection attempts were timing out not only from China, but also from the U.S. The bridge became reachable again from China as soon as it came back online.

- Azadi:4319 (point ⓖ), in contrast, was fully operational at the time of the other bridges' blocking, and the censor nevertheless failed to block it.

We take from the failure to block GreenBelt:5881 and Azadi:5881 that the censor, as late as September 2016, was most likely *not* discovering bridges by inspecting the bridge configuration file in the source code, because if it had been, it would not have missed two of

the bridges in the list. Rather, we suspect that the censor used some kind of network-level analysis—perhaps running a release of Tor Browser in a black-box fashion, and making a record of all addresses it connected to. This would explain why GreenBelt:5881 was not blocked (it couldn't be connected to while the censor was harvesting bridge addresses) and could also explain why Azadi:4319 was not blocked (Tor does not try every bridge simultaneously, so it simply may not have tried to connect to Azadi:4319 in the time the censors allotted for the test). It is consistent with the observation that bridges were not blocked before a release: the censor's discovery process needed a runnable executable.

Azadi:4319 remained unblocked even after an additional port on the same host was blocked in the next release batch. This tidbit will enable us, in the next section, to fairly narrowly locate the onset of bridge discovery based on parsing the bridge configuration file in October 2016.

5.2.7 A switch to blocking before release

The 6.0.6/6.5a4 release batch marked two major changes in the censor's behavior:

1. For the first time, newly added bridges were blocked *before* a release. (Not counting LeifEricson, an old bridge which we had never been able to reach from China.)

2. For the first time, new blocks affected more than one port. (Again not counting LeifEricson.)

The 6.0.6/6.5a4 batch contained eight new bridges. Six were new ports on previously used IP addresses (including LeifEricson:50001, which we expected to be already blocked, but included for completeness). The other two—Lisbeth:443 and NX01:443—were fresh IP addresses. However one of the new bridges, NX01:443, had a twist: we left it commented out in the bridge configuration file, thus:

```
pref(..., "obfs4 192.95.36.142:443 ...");
// Not used yet
// pref(..., "obfs4 85.17.30.79:443 ...");
```

Six of the bridges—all but the exceptional LeifEricson:50000 and NX01:443—were blocked, not quite simultaneously, but within 13 hours of each other (see point ⓗ in Figure 5.2). The blocks happened 14 days (or 22 days in the case of Lisbeth:443 and NX01:443) after ticket merge, and 27 days before the next public release.

We hypothesize that this blocking event indicates a change in the censor's technique, and that in October 2016 the Great Firewall began to discover bridge addresses either by examining newly filed tickets, or by inspecting the bridge configuration file in the source code. A first piece of evidence for the hypothesis is that the bridges were blocked at a time when they were present in the bridge configuration file, but had not yet appeared in a release. The presence of the never-before-seen Lisbeth:443 in the blocked set removes the possibility that the censor spontaneously decided to block additional ports on IP addresses it already knew about, as does the continued reachability of certain blocked bridges on further additional ports.

A second piece of evidence comes from a careful scrutiny of the timelines of the Azadi:4319 and Azadi:6041 bridges. As noted in subsection 5.2.6, Azadi:4316 had unexpectedly been left unblocked in the previous release batch, and it remained so, even after Azadi:6041 was blocked in this batch.

7 September	Azadi:4319 enters the source code
16 September	Azadi:4319 appears in public release 6.0.5
6 October	Azadi:4319 is deleted from the source code, and Azadi:6041 is added
20 October	Azadi:6041 (among others) is blocked
15 November	Azadi:6041 appears in public release 6.0.6

The same ticket that removed Azadi:4319 on October 6 also added Azadi:6041. On October 20 when the bridges were blocked, Azadi:4319 was gone from the bridge configuration file, having been replaced by Azadi:6041. It appears that the yet-unused Azadi:6041 was blocked merely because it appeared in the bridge configuration file, even though it would have been more beneficial to the censor to instead block the existing Azadi:4319, which was still in active use.

The Azadi timeline enables us to locate fairly narrowly the change in bridge discovery techniques. It must have happened during the two weeks between October 6 and October 20, 2016. It cannot have happened before October 6, because at that time Azadi:4319 was still listed, which would have gotten it blocked. And it cannot have happened after October 20, because that is when bridges listed in the file were first blocked.

A third piece of evidence supporting the hypothesis that the censor began to discover bridges through the bridge configuration file is its treatment of the commented-out bridge NX01:443. The bridge was commented out in the 6.0.6/6.5a4 batch, in which it remained unblocked, and uncommented in the following 6.0.8/6.5a6 batch. The bridge was blocked four days after the ticket uncommenting it was merged, which was still 11 days before the public release in which it was to have become active (see point ⓘ in Figure 5.2).

5.2.8 The onset of whole-IP blocking

The blocking event of October 20, 2016 was noteworthy not only because it occurred before a release, but also because it affected more than one port on some bridges. See point ⓗ in Figure 5.2. When GreenBelt:7013 was blocked, so were GreenBelt:5881 (which had escaped blocking in the previous batch) and GreenBelt:12166 (which was awaiting deployment in the next batch). Similarly, when MaBishomarim:7920 and JonbesheSabz:4148 were blocked, so were the Orbot-reserved MaBishomarim:1984 and JonbesheSabz:1984 (point ⓚ), ending an eight-month unblocked streak.

The blocking of Mosaddegh:9332 and Azadi:6041 also affected other ports, though after a delay of some days. We do not have an explanation for why some multiple-port blocks took effect faster than others. The SSH port riemann:22 was blocked at about the same time (point ⓜ), 10 months after the corresponding obfs4 port riemann:443 had been blocked; there had been no changes to the riemann host in all that time. We suspected that the Great Firewall might employ a threshold scheme: once a certain number of individual ports on a particular IP address have been blocked, go ahead and block the entire IP address. But riemann with its single obfs4 port is a counterexample to that idea.

The Great Firewall has been repeatedly documented to block individual ports (or small ranges of ports), for example in 2006 by Clayton et al. [31 §6.1], in 2012 by Winter and Lindskog [199 §4.1], and in 2015 by Ensafi et al. [60 §4.2]. The onset of all-ports blocking is therefore somewhat surprising. Worth nothing, though, is that Wang et al. [189 §7.3], in another test of active probing in May 2017, also found that newly probed bridges became blocked on all ports. The change we saw in October 2016 may therefore be a sign of a more general change in tactics.

This was the first time we saw blocking of multiple ports on bridges that had been introduced during our measurements. LeifEricson may be an example of the same phenomenon happening in the past, before we even began our experiment. The host LeifEricson had, since February 2014, been running bridges on multiple ports, and obfs4 on port 41213 since October 2014. LeifEricson:41213 remained blocked (except intermittently) throughout the entire experiment (see point ⓘ in Figure 5.2). We asked its operator to open additional obfs4 ports so we could rotate through them in successive releases; when we began testing them on August 30, 2016, they were all already blocked. To confirm, on October 4 we asked the operator privately to open additional, randomly selected ports, and they too were blocked, as was the SSH port 22.

In subsection 5.2.5, we observed that ports that had been caught up in whole-IP blocking exhibited different patterns of intermittent reachability after blocking, than did those ports that had been blocked individually. We suspected that a two-tiered system made certain ports double-blocked—blocked both by port and by IP address—which would make their blocking robust to a failure of one of the tiers. The same pattern seems to happen with LeifEricson. The newly opened ports 50000, 50001, and 50002 share brief periods of reachability in September and October 2016, but port 41213 during the same time remained solidly down.

5.2.9 No discovery of Orbot bridges

Orbot, the version of Tor for Android, also includes default bridges. It has its own bridge configuration file, similar to Tor Browser's, but in a different format. Most of Orbot's bridges are borrowed from Tor Browser, so when a bridge gets blocked, it is blocked for users of both Orbot and Tor Browser.

There were, however, a few bridges that were used only by Orbot (see the "Orbot bridges" batch in Figure 5.2). They were only alternate ports on IP addresses that were already used by Tor Browser, but they remained unblocked for over eight months, even as the ports used by Tor Browser were blocked one by one. The Orbot-only bridges were finally blocked—see point ⓚ in Figure 5.2—as a side effect of the whole-IP blocking that began in October 2016 (subsection 5.2.8). (All of the Orbot bridges suffered outages, as Figure 5.2 shows, but they were the result of temporary misconfigurations, not blocking. They were unreachable during those outages from the control site as well.)

These results show that whatever mechanism the censor had for discovering and blocking the default bridges of Tor Browser, it lacked for discovering and blocking those of Orbot. Again we have a case of our assumptions not matching reality—blocking that should be easy to do, and yet is not done. A lesson is that there is a benefit to some degree of compartmentalization between sets of default bridges. Even though they are all, in theory, equally easy to discover, in practice the censor has to build separate automation for each set.

Figure 5.3: Default bridge reachability from a site in Iran. We found no evidence of blocking of default bridges in Iran. What connection failures there were, were also seen from our control site.

5.2.10 Continued blocking of established bridges

We monitored some bridges that were already established and had been distributed before we began our experiments. As expected, they were already blocked at the beginning, and remained so (point ⓘ in Figure 5.2).

5.2.11 No blocking of unused bridges

As a control measure, we reserved a bridge in secret. ndnop4:27668 (see point ⓝ in Figure 5.2) was not published, neither in Tor Browser's bridge configuration file, nor in BridgeDB. As expected, it was never blocked.

5.3 Results from Iran

We had a probe site in Iran from December 2015 to June 2016, a virtual private server, which a personal contact could only provide for us for a limited time.

In contrast to the situation in China, in Iran we found no evidence of blocking. See Figure 5.3. Although there were timeouts and refused connections, they were the result of

failures at the bridge side, as confirmed by a comparison with control measurements. This, despite the fact that Iran is a notorious censor [14], and has in the past blocked Tor directory authorities [7].

It seems that Iran has overlooked the blocking of default bridges. Tor Metrics shows thousands of simultaneous bridge users in Iran since 2014 [178], so it is unlikely that the bridges were simply blocked in a way that our probing script could not detect. However, in Kazakhstan we did find such situation, with bridges being effectively blocked despite the firewall allowing TCP connections to them.

5.4 Results from Kazakhstan

We had a single probe site in Kazakhstan between December 2016 and May 2017. It was a VPN node with IP address 185.120.77.110 . It was in AS 203087, which belongs to GoHost.kz, a Kazakh hosting provider. The flaky VPN connection left us with two extended gaps in measurements.

The bridge blocking in Kazakhstan had a different nature than that which we observed in China. Refer to Figure 5.4: every measurement agreed with the control site, with the sole exception of LeifEricson:41213 (not shown), which was blocked as it had been in China. However there had been reports of the blocking of Tor and pluggable transports since June 2016 [88 §obfs blocking]. The reports stated that the TCP handshake would succeed, but the connection would stall (with no packets received from the bridge) a short time after the connection was underway.

We deployed an additional probing script in Kazakhstan. This one tried not only to make a TCP connection, but also establish a full obfs4 connection and build a Tor circuit. Tor reports its connection progress as a "bootstrap" percentage: progression from 0% to 100% involves first making an obfs4 connection, then downloading directory information and the consensus, and finally building a circuit. Figure 5.5 shows the results of the tests. What we found was consistent with reports: despite being reachable at the TCP layer, some bridges would fail bootstrapping at 10% (e.g., Mosaddegh:80 and GreenBelt:80) or 25% (e.g., Mosaddegh:443 and GreenBelt:443). For three of the bridges (Mosaddegh:9332, Lisbeth:443, and NX01:443) we caught the approximate moment of blocking. Initially they bootstrapped to 100% and agreed with the control, but later they reached only 25% and disagreed with the control. Incidentally, these results suggest that Kazakhstan, too, blocks on a per-port basis, because for a time Mosaddegh:80 and Mosaddegh:443 were blocked while Mosaddegh:9332 was unblocked. Two more bridges (cymrubridge31:80 and cymrubridge33:80) remained unblocked.

ndnop3:10527 and ndnop5:13764, in the 5.5/6.0a1 batch, are a special case. Their varying bootstrap percentages were caused by a misconfiguration on the bridge itself (a file descriptor limit was set too low). Even from the control site in the U.S., connections would fail to bootstrap to 100% about 35% of the time. Still, it appears that both bridges were also blocked in Kazakhstan, because from the control site the bootstrap percentage would oscillate between 10% and 100%; while from Kazakhstan it would oscillate between 10% to 25%.

The bridges in the 6.0.6/6.5a4 and 6.0.8/6.5a6 batches were blocked on or around January 26, 2017. This sets the blocking delay at either 71 or 43 days after public release, depending on which release you compare against.

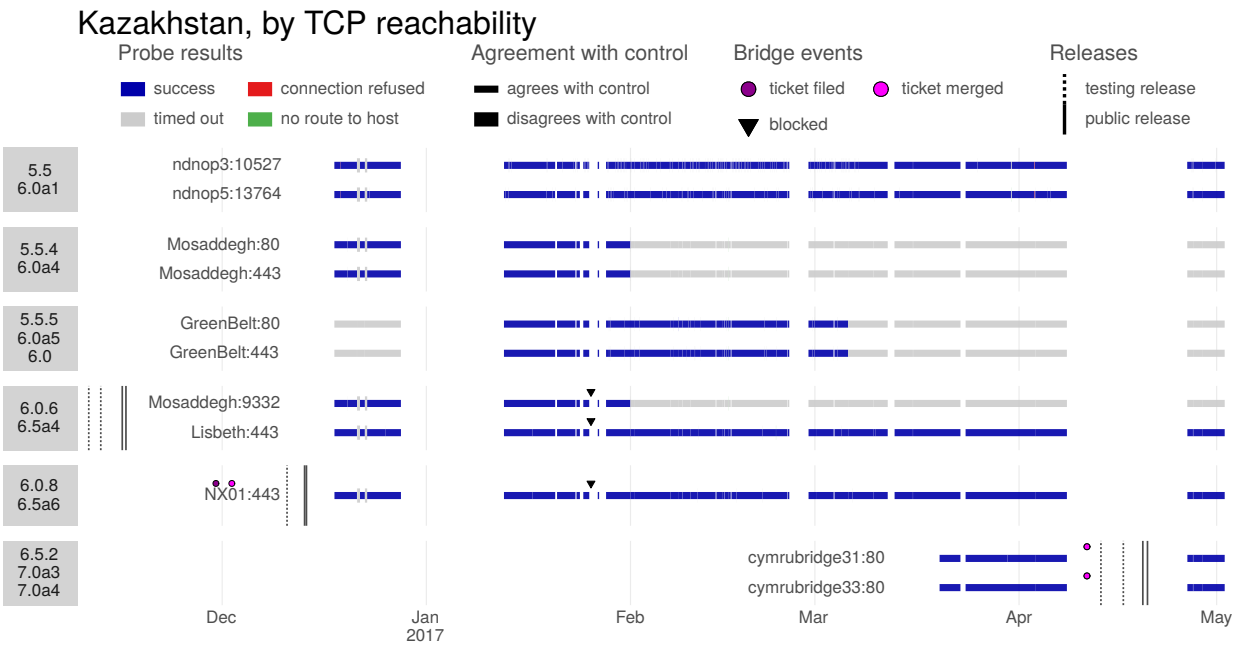

Figure 5.4: Default bridge reachability from a site in Kazakhstan. Judging by TCP reachability alone, it would seem that there is no disagreement with the control site—and therefore no blocked bridges However, the more intensive experiment of Figure 5.5, below, reveals that despite being reachable at the TCP layer, most of the bridges were in fact effectively blocked.

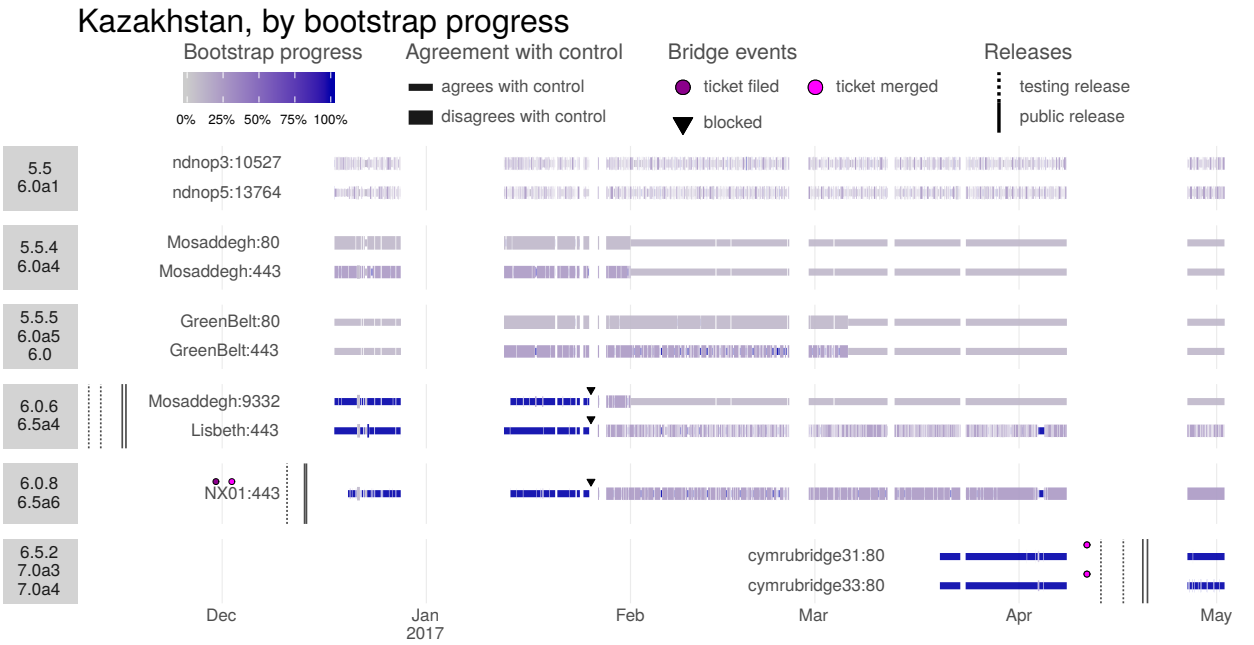

Figure 5.5: Default bridge bootstrap progress from a site in Kazakhstan. In contrast to Figure 5.4, above, this experiment built a full obfs4 connection and Tor circuit, revealing blocking beyond the TCP handshake. Tor reports its connection progress as a percentage; so here, "success" is on a continuum from 0% to 100%, as is the degree of agreement with the control site. The first three batches were blocked since before we started measuring; the next two were blocked in January, and the last was not blocked.

Chapter 6

Domain fronting

Domain fronting is a general-purpose circumvention technique based on HTTPS. It disguises the true destination of a client's messages by routing them through a large web server or content delivery network that hosts many web sites. From the censor's point of view, messages appear to go not to their actual (presumably blocked) destination, but to some other *front domain*, one whose blocking would result in high collateral damage. Because (with certain caveats) the censor cannot distinguish domain-fronted HTTPS requests from ordinary HTTPS requests, it cannot block circumvention without also blocking the front domain. Domain fronting primarily addresses the problem of detection by address (Section 2.3), but also deals with detection by content (Section 2.2) and active probing (Chapter 4). Domain fronting is today an important component of many circumvention systems.

The core idea of domain fronting is the use of different domain names at different protocol layers. When you make an HTTPS request, the domain name of the server you're trying to access normally appears in three places that are visible to the censor:

- the DNS query
- the client's TLS Server Name Indication (SNI) extension [59 §3]
- the server's TLS certificate [42 §7.4.2]

and in one place that is not visible to the censor, because it is encrypted:

- the HTTP Host header [65 §5.4]

In a normal request, the same domain name appears in all four places, and all of them except for the Host header afford the censor an easy basis for blocking. The difference in a domain-fronted request is that the domain name in the Host header, on the "inside" of the request, is not the same as the domain that appears in the other places, on the "outside." Figure 6.1 shows the first steps of a client making a domain-fronted request.

The SNI extension and the Host header serve similar purposes. They both enable virtual hosting, which is when one server handles requests for multiple domains. Both fields allow the client to tell the server which domain it wants to access, but they work at different layers. The SNI works at the TLS layer, telling the server which certificate to send. The Host header works at the HTTP layer, telling the server what contents to serve. It is something of an

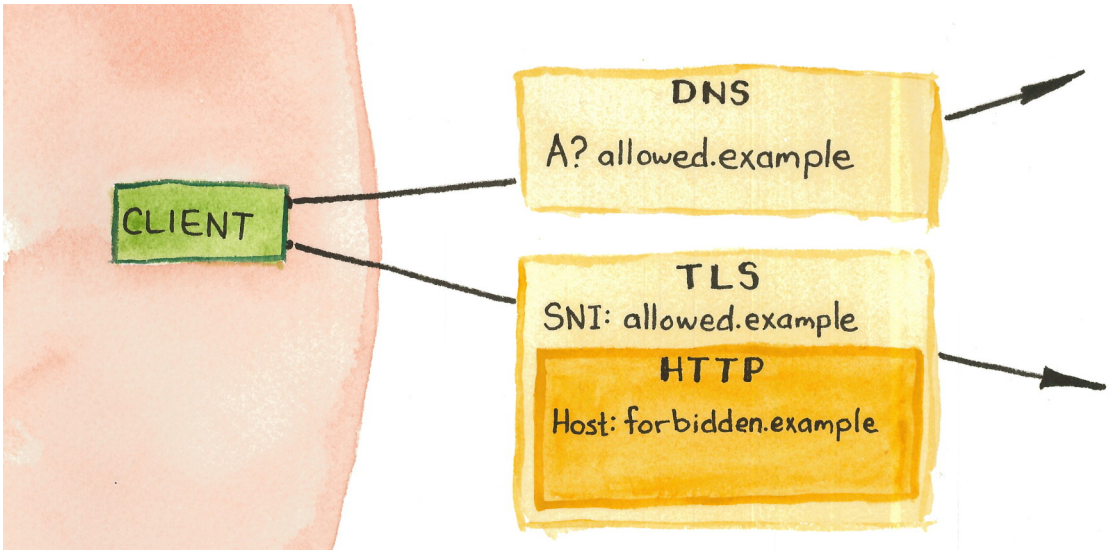

Figure 6.1: Domain fronting uses different names at different protocol layers. The forbidden destination domain is encrypted within the TLS layer. The censor sees only a front domain, one chosen to be expensive to block. Not shown here, the server's certificate will also expose only the front domain, because the certificate is a property of the TLS layer, not the HTTP layer.

accident that these two partially redundant fields both exist. Before TLS, virtual hosting required only the Host header. The addition of TLS creates a chicken-and-egg problem: the client cannot send the Host header until the TLS handshake is complete, and the server cannot complete the TLS handshake without knowing which certificate to send. The SNI extension resolves the deadlock by sending the domain name in plaintext in the TLS layer. Domain fronting takes advantage of decoupling the two normally coupled values. It relies on the server decrypting the TLS layer and throwing it away, then routing requests according to the Host header.

Virtual hosting, in the form of content delivery networks (CDNs), is now common. A CDN works by placing an "edge server" between the client and the destination, called an "origin server" in this context. When the edge server receives an HTTP request, it forwards the request to the origin server named by the Host header. The edge server receives the response from the origin server and forwards it back to the client. The edge server is effectively a proxy: the client never contacts the destination directly, but only through the intermediary CDN, which foils address-based blocking of the destination the censor may have imposed. Domain fronting also works on application hosting services like Google App Engine, because one can upload a simple application that emulates a CDN. The contents of the client's messages, as well as the domain name of the true destination, are protected by TLS encryption. The censor may, in an attempt to block domain fronting, block CDN edge servers or the front domain, but only at the cost of blocking all other, non-circumvention-related traffic to those addresses, with whatever collateral damage that entails.

Domain fronting may be an atypical use of HTTPS, but it is not a way to get free CDN service. A CDN does not forward requests to arbitrary domains, only to domains belonging to one of its customers. Setting up domain fronting requires becoming a customer of a CDN and paying for service—and the cost can be high, as Section 6.3 shows.

It may seem at first that domain fronting is only useful for accessing HTTPS web sites, and then only when they are hosted on a CDN. But extending the idea to work with arbitrary destinations only requires the minor additional step of running an HTTPS-based proxy server and hosting it on the web service in question. The CDN forwards to the proxy, which then forwards to the destination. Domain fronting shields the address of the proxy, which does not pose enough risk of collateral damage, on its own, to resist blocking. Exactly this sort of HTTPS tunneling underlies meek, a circumvention system based on domain fronting that is discussed further in Section 6.2.

One of the best features of domain fronting is that it does not require any secret information, completely bypassing the proxy distribution problem (Section 2.3). The address of the CDN edge server, the address of the proxy hidden behind it, the fact that some fraction of traffic to the edge server is circumvention—all of these may be known by the censor, without diminishing the system's blocking resistance. This is not to say, of course, that domain fronting is impossible to block—as always, a censor's capacity to block depends on its tolerance for collateral damage. But the lack of secrecy makes the censor's choice stark: allow circumvention, or block a domain. This is the way to think about circumvention in general: not "can it be blocked?" but "what does it cost to block?"

6.1 Work related to domain fronting

I did not invent domain fronting. I did, however, give it a name, help popularize its use, and produce an important implementation. As far as I have been able to find out, the first implementation of domain fronting was in GoAgent, a circumvention system, circa 2012. GoAgent employed a variant of fronting where the SNI is omitted, rather than being faked. Earlier in 2012, Bryce Boe wrote a blog post [17] outlining how to use Google App Engine as a proxy, and suggested that sending a false SNI could bypass SNI whitelisting. Even farther back, in 2004, when HTTPS and CDNs were less common, Köpsell and Hillig [118 §5.2] foresaw the possibilities of a situation such as exists today: "Imagine that all web pages of the United States are only retrievable (from abroad) by sending encrypted requests to one and only one special node. Clearly this idea belongs to the 'all or nothing' concept because a blocker has to block all requests to this node."

Refraction networking is the name for a class of circumvention techniques that share similarities with domain fronting. The idea was introduced in 2011 with the designs Cirripede [104], CurveBall [112], and Telex [203]. In refraction networking, it is network routers that act as proxies, lying at the middle of network paths rather than at the ends. The client "tags" its messages in a way that the censor cannot detect (analogously to the way the Host header is encrypted in domain fronting). When the router finds a tagged message, it shunts the message away from its nominal destination and towards some other, covert destination. Refraction networking derives its blocking resistance from the collateral damage that would result from blocking the cover channel (typically TLS) or the refraction-capable network routers. Refraction networking has the potential to be the basis of exceptionally high-performance circumvention, as a test deployment in Spring 2017 demonstrated [94].

CloudTransport [23], proposed in 2014, is like domain fronting in many respects. It uses HTTPS to a shared server (in this case a cloud storage server). The specific storage area

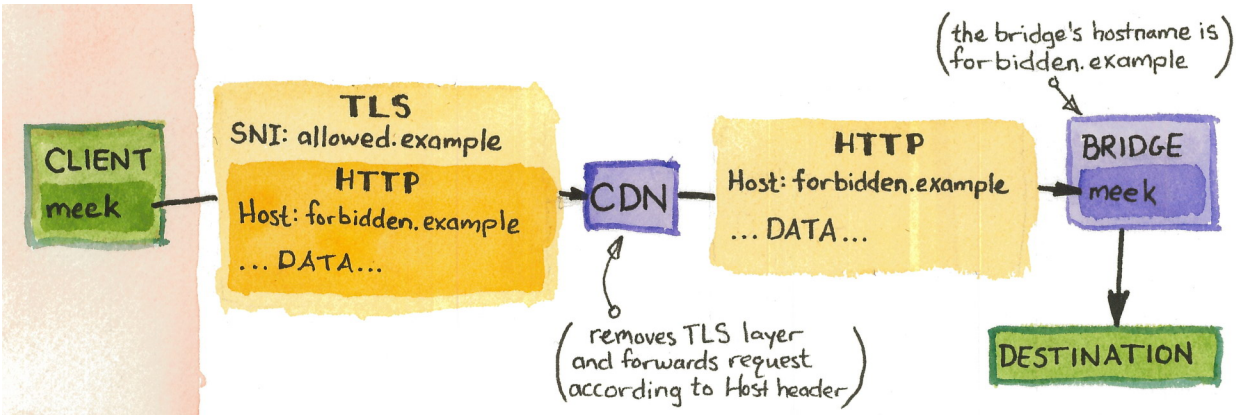

Figure 6.2: Putting it together: how to build a circumvention system around domain fronting. The CDN acts as a single-purpose proxy, only capable of forwarding to destinations within its own network—one of which is a bridge, which we control. The bridge acts as a general-purpose proxy, capable of reaching any destination. Fronting through the CDN hides the bridge's address, which the censor would otherwise block.

being accessed—what the censor would like to know—is encrypted, so the censor cannot block CloudTransport without blocking the storage service completely.

In 2015 I published a paper on domain fronting [89] with Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. In it, we described the experience of deploying domain fronting on Tor, Lantern [119], and Psiphon [157], and began an investigation into side channels, such as packet size and timing, that a censor might use to detect domain fronting. The Tor deployment, called meek, is the subject of Sections 6.2 and 6.3.

Later in 2015 there were a couple of papers on the detection of circumvention transports, including meek. Tan et al. [174] measured the Kullback–Leibler divergence between the distributions of packet size and packet timing in different protocols. (The paper is written in Chinese and my understanding of it is based on an imperfect translation.) Wang et al. [186] built classifiers for meek among other protocols using entropy, timing, and transport-layer features. They emphasized practical classifiers and tested their misclassification rates against real traffic traces.

6.2 A pluggable transport for Tor

I am the main author and maintainer of meek, a pluggable transport for Tor based on domain fronting. meek uses domain-fronted HTTP POST requests as the primitive operation to send or receive chunks of data up to a few kilobytes in size. The intermediate CDN receives domain-fronted requests and forwards them to a Tor bridge. Auxiliary programs on the client and the bridge convert the sequence of HTTP requests to the byte stream expected by Tor. The Tor processes at either end are oblivious to the domain-fronted that is going on between them. Figure 6.2 shows how the components and protocol layers interact.

When the client has something to send, it issues a POST request with data in the body; the server sends data back in the body of its responses. HTTP/1.1 does not provide a way for a server to preemptively push data to a client, so the meek server buffers its outgoing

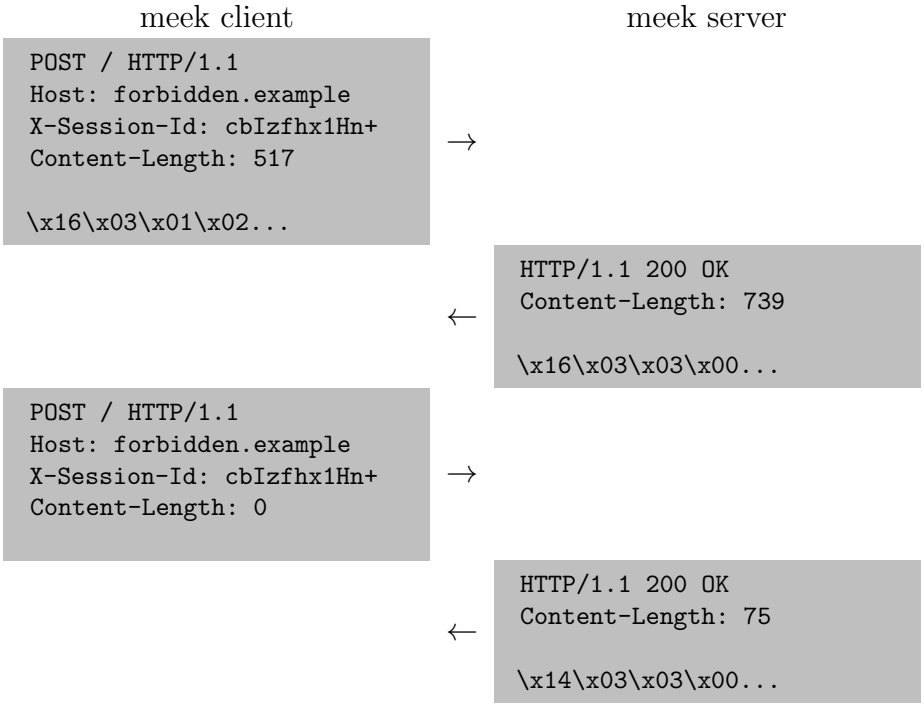

Figure 6.3: The HTTP-based framing protocol of meek. Each request and response is domain-fronted. The second POST is an example of an empty polling request, sent only to give the server an opportunity to send data downstream.

data until it receives a request, then includes the buffered data in the body of the HTTP response. The client must poll the server periodically, even when it has nothing to send, to give the server an opportunity to send back whatever buffered data it may have. The meek server must handle multiple simultaneous clients. Each client, at the beginning of a session, generates a random session identifier string and sends it with its requests in a special X-Session-Id HTTP header. The server maintains separate connections to the local Tor process for each session identifier. Figure 6.3 shows a sequence of requests and responses.

Even with domain fronting to hide the destination request, a censor may try to distinguish circumventing HTTPS connections by their TLS fingerprint. TLS implementations have a lot of latitude in composing their handshake messages, enough that it is possible to distinguish different TLS implementations through passive observation. For example, the Great Firewall used Tor's TLS fingerprint for detection as early as 2011 [48]. For this reason, meek strives to make its TLS fingerprint look like that of a browser. It does this by relaying its HTTPS requests through a local headless browser (which is completely separate from the browser that the user interacts with).

meek first appeared in Tor Browser in October 2014 [153], and continues in operation to the present. It is Tor's second-most-used transport (behind obfs4) [176]. The next section is a detailed history of its deployment.

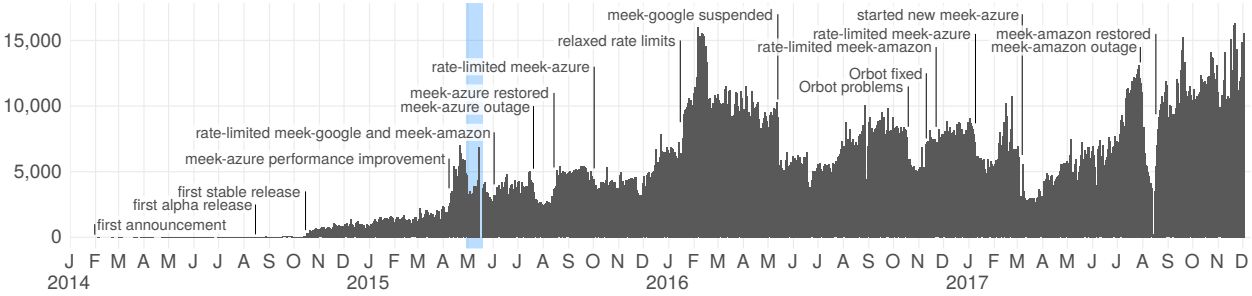

Figure 6.4: Estimated mean number of concurrent users of the meek pluggable transport, with selected events. This graph is an updated version of Figure 5 from the 2015 paper "Blocking-resistant communication through domain fronting" [89]; the vertical blue stripe divides old and new data. The user counts come from Tor Metrics.

	Google	Amazon	Azure	total
2014 Jan	$0.00	—	—	$0.00
Feb	$0.09	—	—	$0.09
Mar	$0.00	—	—	$0.00
Apr	$0.73	—	—	$0.73
May	$0.69	—	—	$0.69
Jun	$0.65	—	—	$0.65
Jul	$0.56	$0.00	—	$0.56
Aug	$1.56	$3.10	—	$4.66
Sep	$4.02	$4.59	$0.00	$8.61
Oct	$40.85	$130.29	$0.00	$171.14
Nov	$224.67	$362.60	$0.00	$587.27
Dec	$326.81	$417.31	$0.00	$744.12
2014 total	$600.63	$917.89	$0.00	$1,518.52

	Google	Amazon	Azure	total
2016 Jan	$771.17	$1,581.88	$329.10	$2,682.15
Feb	$986.39	$977.85	$445.83	$2,410.07
Mar	$1,079.49	$865.06	$534.71	$2,479.26
Apr	$1,169.23	$1,074.25	$508.93	$2,752.41
May	$525.46	$1,097.46	$513.56	$2,136.48
Jun	—	$1,117.67	$575.50	$1,693.17
Jul	—	$1,121.71	$592.47	$1,714.18
Aug	—	$1,038.62	$607.13	$1,645.75
Sep	—	$932.22	$592.92	$1,525.14
Oct	—	$1,259.19	$646.00	$1,905.19
Nov	—	$1,613.00	$597.76	$2,210.76
Dec	—	$1,569.84	$1,416.10	$2,985.94
2016 total	$4,531.74	$14,248.75	$7,360.01	$26,140.50

	Google	Amazon	Azure	total
2015 Jan	$464.37	$669.02	$0.00	$1,133.39
Feb	$650.53	$604.83	$0.00	$1,255.36
Mar	$690.29	$815.68	$0.00	$1,505.97
Apr	$886.43	$785.37	$0.00	$1,671.80
May	$871.64	$896.39	$0.00	$1,768.03
Jun	$601.83	$820.00	$0.00	$1,421.83
Jul	$732.01	$837.08	$0.00	$1,569.09
Aug	$656.76	$819.59	$154.89	$1,631.24
Sep	$617.08	$710.75	$490.58	$1,818.41
Oct	$672.01	$110.72	$300.64	$1,083.37
Nov	$602.35	$474.13	$174.18	$1,250.66
Dec	$561.29	$603.27	$172.60	$1,337.16
2015 total	$8,006.59	$8,146.83	$1,292.89	$17,446.31

	Google	Amazon	Azure	total
2017 Jan	—	$1,550.19	$1,196.28	$2,746.47
Feb	—	$1,454.68	$960.01	$2,414.69
Mar	—	$2,298.75	?	$2,298.75+
Apr	—	?	?	?
May	—	?	?	?
Jun	—	?	?	?
Jul	—	?	?	?
Aug	—	?	?	?
Sep	—	?	?	?
Oct	—	?	?	?
Nov	—	?	?	?
2017 total	—	$5,303.62+	$2,156.29+	$7,459.91+

grand total	$13,138.96	$28,617.09+	$10,809.19+	$52,565.24+

Table 6.5: Costs for running meek, compiled from my monthly reports [137 §Costs]. (The reference has minor arithmetic errors that are corrected here.) meek ran on three different web services: Google App Engine, Amazon CloudFront, and Microsoft Azure. The notation '—' means meek wasn't deployed on that service in that month; for example, we stopped using App Engine after May 2016 following the suspension of the service (see discussion on page 58). The notation '?' marks the months after I stopped handling the invoices personally. I don't know the costs for those months, so certain totals are marked with '+' to indicate that they are higher than the values shown.

6.3 An unvarnished history of meek deployment

Fielding a circumvention system and keeping it running is full of unexpected challenges. At the time of the publication of the domain fronting paper [89] in 2015, meek had been deployed for only a year and a half. Here I will recount the history of the project from its inception to the present, a period of four years. As the main developer and project leader, I have a unique perspective that I hope to share. As backdrops to the narrative, Figure 6.4 shows the estimated concurrent number of users of meek over its existence, and Table 6.5 shows the monthly cost to run it.

2013: Precursors; prototypes

The prehistory of meek begins in 2013 with flash proxy [84], a circumvention system built around web browser–based proxies. Flash proxy clients need a secure rendezvous, a way to register their address to a central facilitator, so that flash proxies may connect back to them. Initially there were only two means of registration: flashproxy-reg-http, which sent client registrations as HTTP requests; and flashproxy-reg-email, which sent client registrations to a distinguished email address. We knew that flashproxy-reg-http was easily blockable; flashproxy-reg-email had good blocking resistance but was somewhat slow and complicated, requiring a server to poll for new messages. At some point, Jacob Appelbaum showed me an example of using domain fronting—though we didn't have a name for it then—to access a simple HTML-rewriting proxy based on Google App Engine. I eventually realized that the same trick would work for flash proxy rendezvous. I proposed a design [21] in May 2013 and within a month Arlo Breault had written flashproxy-reg-appspot, which worked just like flashproxy-reg-http, except that it fronted through www.google.com rather than contacting the registration server directly. The fronting-based registration became flash proxy's preferred registration method, being faster and simpler than the email-based one.

The development of domain fronting, from a simple rendezvous technique to a full-fledged bidirectional transport, seems slow in retrospect. All the pieces were there; it was a matter of putting them together. I did not immediately appreciate the potential of domain fronting when I first saw it. Even after the introduction of flashproxy-reg-appspot, months passed before the beginning of meek. The whole idea behind flash proxy rendezvous is that the registration channel can be of low quality—unidirectional, low-bandwidth, and high-latency—because it is only used to bootstrap into a more capable channel (WebSocket, in flash proxy's case). Email fits this model well: not good for a general-purpose channel, but just good enough for rendezvous. The fronting-based HTTP channel, however, was more capable than needed for rendezvous, being bidirectional and reasonably high-performance. Rather than handing off the client to a flash proxy, it should be possible to carry all the client's traffic through the same domain-fronted channel. It was around this time that I first became aware of the circumvention system GoAgent through the "Collateral Freedom" [163] report of Robinson et al. GoAgent used an early form of domain fronting, issuing HTTP requests directly from a Google App Engine server. According to the report, GoAgent was the most used circumvention tool among a group of users in China. I read the source code of GoAgent in October 2013 and wrote ideas about writing a similar pluggable transport [73], which would become meek.

I dithered for a while over what to call the system I was developing. Naming things is the worst part of software engineering. My main criteria were that the name should not sound macho, and that it should be easier to pronounce than "obfs." I was self-conscious that the idea at the core of the system, domain fronting was a simple one and easy to implement. Not wanting to oversell it, I settled on the name "meek," in small letters for extra meekness.

I lost time in the premature optimization of meek's network performance. I was thinking about the request–response nature of HTTP, and how requests and responses could conceivably arrive out of order (even if reordering was unlikely to occur in practice, because of keepalive connections and HTTP pipelining). I made several attempts at a TCP-like reliability and sequencing layer, none of which were satisfactory. I wrote a simplified experimental prototype called "meeker," which simply prepended an HTTP header before the client and server streams, but meeker only worked for direct connections, not through an HTTP-aware intermediary like App Engine. When I explained these difficulties to George Kadianakis in December 2013, he advised me to forget the complexity and implement the simplest thing that could work, which was good advice. I started implementing a version that strictly serialized HTTP requests and responses.

2014: Development; collaboration; deployment

According to the Git revision history, I started working on the source code of meek proper on January 26, 2014. I made the first public announcement on January 31, 2014, in a post to the tor-dev mailing list titled "A simple HTTP transport and big ideas" [66]. (If the development time seems short, it's only because months of prototypes and false starts cleared the way.) In the post, I linked to the source code, described the protocol, and explained how to try it, using an App Engine instance I set up shortly before. At this time there was no web browser TLS camouflage, and only App Engine was supported. I was not yet using the term "domain fronting." The big ideas of the title were as follows: we could run one big public bridge rather than relying on multiple smaller bridges as other transports did; a web server with a PHP "reflector" script could take the place of a CDN, providing a diversity of access points even without domain fronting; we could combine meek with authentication and serve a 404 to unauthenticated users; and Cloudflare and other CDNs are alternatives to App Engine. We did end up running a public bridge for public benefit (and later worrying over how to pay for it), and deploying on platforms other than App Engine (with Tor we use other CDNs, but not Cloudflare specifically). Arlo Breault would write a PHP reflector, though there was never a repository of public meek reflectors as there were for other types of Tor bridges. Combining meek with authentication never happened; it was never needed for our public domain-fronted instances because active probing doesn't help the censor in those cases anyway.

During the spring 2014 semester (January–May) I was enrolled in Vern Paxson's Internet/Network Security course along with fellow student Chang Lan. We made the development and security evaluation of meek our course project. During this time we built browser TLS camouflage extensions, tested and polished the code, and ran performance tests. Our final report, "Blocking-resistant communication through high-value web services," became the kernel of our later research paper.

I began the process of getting meek integrated into Tor Browser in February 2014 [85].

The initial integration would be completed in August 2014. In the intervening time, along with much testing and debugging, Chang Lan and I wrote browser extensions for Chrome and Firefox in order to hide the TLS fingerprint of the base meek client. I placed meek's code in the public domain (Creative Commons CC0 [34]) on February 8, 2014. The choice of (non-)license was a strategic decision to encourage adoption by projects other than Tor.

In March 2014, I met some developers of Lantern at a one-day hackathon sponsored by OpenITP [24]. Lantern developer Percy Wegmann and I realized that the meek code I had been working on could act as a glue layer between Tor and the HTTP proxy exposed by Lantern, in effect allowing you to use Lantern as a pluggable transport for Tor. We worked out a prototype and wrote a summary of the process [75]. In that specific application, we used meek not for its domain-fronting properties but for its HTTP-tunneling properties; but the early contact with other circumvention developers was valuable.

June 2014 brought a surprise: the Great Firewall of China blocked all Google services [4, 96]. It would be vain to think that it was in response to the nascent deployment of meek on App Engine; a much more likely cause was Google's decision to begin using HTTPS for web searches, which would foil keyword-based URL filtering. Nevertheless, the blocking cast doubt on the feasibility of domain fronting: I had believed that blocking all of Google would be too costly in terms of collateral damage to be sustained for long by any censor, even the Great Firewall, and that belief was wrong. In any case, we now needed fronts other than Google in order to have any claim of effective circumvention in China. I set up additional backends: Amazon CloudFront and Microsoft Azure. When meek made its debut in Tor Browser, it would offer three modes: meek-google, meek-amazon, and meek-azure.

Google sponsored a summit of circumvention researchers in June 2014, at which I presented domain fronting. (By this time I had started using the term "domain fronting," realizing that what I had been working on needed a specific name. I have tried to the idea "domain fronting" separate from the implementation "meek," but the two terms have sometimes gotten confused.) Developers from Lantern and Psiphon where there—I was pleased to learn that Psiphon had already implemented and deployed domain fronting after reading my mailing list posts. The meeting started a fruitful collaboration between the developers of Tor, Lantern, and Psiphon.

Chang, Vern, and I submitted a paper on domain fronting to the Network and Distributed System Security Symposium in August 2014, whence it was rejected. One reviewer said the technique was already well known; the others generally wanted to see more on the experience of deployment, and a deeper investigation into resistance against traffic analysis attacks based on packet sizes and timing.

The first public release of Tor Browser that had a built-in easy-to-use meek client was version 4.0-alpha-1 on August 12, 2014 [29]. This was an alpha release, used by fewer users than the stable release. I made a blog post explaining how to use it a few days later [74]. The release and blog post had a positive effect on the number of users, however the absolute numbers from around this time are uncertain, because of a mistake I made in configuring the meek bridge. I was running the meek bridge and the flash proxy bridge on the same instance of Tor; and because of how Tor's statistics are aggregated, the counts of the two transports were spuriously correlated [78]. I switched the meek bridge to a separate instance of Tor on September 15; numbers after that date are more trustworthy. In any case, the usage before this first release was tiny: the App Engine bill, at a rate of $0.12/GB with one GB free each

day, was less than $1.00 per month for the first seven months of 2014 [137 §Costs]. In August, the cost began to be nonzero every day, and would continue to rise from there. See Table 6.5 on page 52 for a history of monthly costs.

Tor Browser 4.0 [153] was released on October 15, 2014. It was the first stable (not alpha) release to have meek, and it had an immediate effect on the number of users: which jumped from 50 to 500 within a week. (The increase was partially conflated with a failure of the meek-amazon bridge to publish statistics before that date, but the other bridge, servicing both meek-google and meek-azure, individually showed the same increase.) It was a lesson in user behavior: although meek had been available in an alpha release for two months already, evidently a large number of users did not know of it or chose not to try it until the first stable release. At that time, the other transports available were obfs3, FTE, ScrambleSuit, and flash proxy.

2015: Growth; restraints; outages

Through the first part of 2015, the estimated number of simultaneous users continued to grow, reaching about 2,000, as we fixed bugs and Tor Browser had further releases. The first release of Orbot that included meek appeared in February [93].

We submitted a revised version of the domain fronting paper [89], now with contributions from Psiphon and Lantern, to the Privacy Enhancing Technologies Symposium, where it was accepted and appeared on June 30 at the symposium.

The increasing use of domain fronting by various circumvention tools began to attract more attention. A March 2015 article by Eva Dou and Alistair Barr in *The Wall Street Journal* [53] described domain fronting and "collateral freedom" in general, depicting cloud service providers as being caught in the crossfire between censors and circumventors. The journalists contacted me but I declined to be interviewed; I thought it was not the right time for extra publicity, and anyway personally did not want to deal with doing an interview. Shortly thereafter, GreatFire, an anticensorship organization that was mentioned in the article, experienced a new type of denial-of-service attack [171], caused by a Chinese network attack system later known as the Great Cannon [129]. They blamed the attack on the attention brought by the news article. As further fallout, Cloudflare, a CDN which Lantern used for fronting and whose CEO was quoted in the article, stopped supporting domain fronting [155], by beginning to enforce a match between the SNI and the Host header

Since its first deployment, the Azure backend had been slower, with fewer users, than the other two options, App Engine and CloudFront. For months I had chalked it up to limitations of the platform. In April 2015, though, I found the real source of the problem: the component I wrote that runs on Azure, receives domain-fronted HTTP requests and forwards them to the meek bridge, was not reusing TCP connections. For every outgoing request, the code was doing a fresh TCP and TLS handshake—causing a bottleneck at the bridge as its CPU tried to cope with all the incoming TLS. When I fixed the code to reuse connections [67], the number of users (overall, not only for Azure) had a sudden jump, increasing from 2,000 to reaching 6,000 in two weeks. Evidently, we had been leaving users on the table by having one of the backends not run as fast as possible.

The deployment of domain fronting was being partly supported by a $500/month grant from Google. Already in February 2015, the monthly cost for App Engine alone began to

exceed that amount [137 §Costs]. In an effort to control costs, in May 2015 we began to rate-limit the App Engine and CloudFront bridges, deliberately slowing the service so that fewer would use it. Until October 2015, the Azure bridge was on a research grant provided by Microsoft, so we allowed it to run as fast as possible. When the grant expired, we rate-limited the Azure bridge as well. This rate-limiting explains the relative flatness of the user graph from May to the end of 2015.

Google changed the terms of service governing App Engine in 2015. (I received a message announcing the change in May, but it seems the changes had been changed online since March.) The updated terms included a paragraph that seemed to prohibit running a proxy service [97]:

> *Networking.* Customer will not, and will not allow third parties under its control to: (i) use the Services to provide a service, Application, or functionality of network transport or transmission (including, but not limited to, IP transit, virtual private networks, or content delivery networks); or (ii) sell bandwidth from the Services.

This was a stressful time: we seemed to have Google's support, but the terms of service said otherwise. I contacted Google to ask for clarification or guidance, in the meantime leaving meek-google running; however I never got an answer to my questions. The point became moot a year later, when Google shut down our App Engine project, for another reason altogether; see below.

By this time we had not received reports of any attempts to block domain fronting. We did, however, suffer a few accidental outages (which are just as bad as blocking, from a client's point of view). Between July 20 and August 14, an account transition error left the Azure configuration broken [77]. I set up another configuration on Azure and published instructions on how to use it, but it would not be available to the majority of users until the next release of Tor Browser, which happened on August 11. Between September 30 and October 9, the CloudFront bridge was effectively down because of an expired TLS certificate. When it rebooted on October 9, an administrative oversight caused its Tor relay identity fingerprint to change—meaning that clients expecting the former fingerprint refused to connect to it [87]. The situation was not fully resolved until November 4 with the next release of Tor Browser: cascading failures led to over a month of downtime.

In October 2015 there appeared a couple of research papers that investigated meek's susceptibility to detection via side channels. Tan et al. [174] used Kullback–Leibler divergence to quantify the differences between protocols, with respect to packet size and interarrival time distributions. Their paper is written in Chinese; I read it in machine translation. Wang et al. [186] published a more comprehensive report on detecting meek (and other protocols), emphasizing practicality and precision. They showed that some previously proposed classifiers would have untenable false-positive rates, and constructed a classifier for meek based on entropy and timing features. It's worth noting that since the first reported efforts to block meek in 2016, censors have preferred, as far as we can tell, to use techniques other than those described in these papers.

A side benefit of building a circumvention system atop Tor is easy integration with Tor Metrics—the source of the user number estimates in this section. Since the beginning of meek's deployment, we had known about a problem with the way it integrates with Tor

Metrics. Tor pluggable transports geolocate the client's IP address in order to aggregate statistics by country. But when a meek bridge receives a connection, the "client IP address" it sees is not that of the true client, but rather that of some cloud server, the intermediary through which the client's domain-fronted traffic passes. So the total user counts were fine, but the per-country counts were meaningless. For example, because App Engine's servers were located in the U.S., every meek-google connection was being counted as if it belonged to a client in the U.S. By the end of 2015, meek users were a large enough fraction (about 20%) of all bridge users that they were skewing the overall per-country counts. I wrote a patch [90] to have the client's true IP address forwarded through the network intermediary in a special HTTP header, which fixed the per-country counts from then on.

2016: Taking off the reins; misuse; blocking efforts

In mid-January 2016 the Tor Project asked me to raise the rate limits on the meek bridges, in anticipation of rumored attempts to block Tor in Egypt. I asked the bridge operators raise the limits from approximately 1 MB/s to 3 MB/s. The effect of the relaxed rate limits was immediate: the count shot up as high 15,000 simultaneous users, briefly making meek Tor's most-used pluggable transport, before settling in at around 10,000.

The first action that may have been a deliberate attempt to block domain fronting came on January 29, 2016, when the Great Firewall of China blocked one of the edge servers of the Azure CDN. The blocking was by IP address, a severe method: not only the domain name we were using for fronting, but thousands of other names became inaccessible. The block lasted about four days. On February 2, the server changed its IP address (simply incrementing the final octet from .200 to .201), causing it to become unblocked. I am aware of no other incidents of edge server blocking.

The next surprise was on May 13, 2016. meek's App Engine backend stopped working and I got a notice:

> We've recently detected some activity on your Google Cloud Platform/API Project ID meek-reflect that appears to violate our Terms of Service. Please take a moment to review the Google Cloud Platform Terms of Service or the applicable Terms of Service for the specific Google API you are using.
>
> Your project is being suspended for committing a general terms of service violation.
>
> We will delete your project unless you correct the violation by filling in the appeals form available on the project page of Developers Console to get in touch with our team so that we can provide you with more details.

My first thought—which turned out to be wrong—was that it was because of the changes to the terms of service that had been announced the previous year. I tried repeatedly to contact Google and learn the nature of the violation, But none of my inquiries received even an acknowledgement. It was not until June 18 that I got some insight, through an unofficial channel, about what happened. Some botnet had apparently been misusing meek for command and control purposes. Its operators had not even bothered to set up their own App Engine project; they were free-riding on the service we had been operating for the public. Although we may have been able to reinstate the meek-google service, seeing as the

suspension was the result of someone else's actions, not ours, with the existing uncertainty around the terms of service I didn't have the heart to pursue it. meek-google remained off, and users migrated to meek-amazon or meek-azure. It turned out, later, that it had been no common botnet misusing meek-google, but an organized political hacker group, known as Cozy Bear or APT29. The group's malware would install a backdoor that operated over a Tor onion service, and used meek for camouflage. Dunwoody and Carr presented these findings at DerbyCon in September 2016 [56], and in a blog post [55] in March 2017 (which is where I learned of it).

The year 2016 brought the first reports of efforts to block meek. These efforts all had in common that they used TLS fingerprinting in conjunction with SNI inspection. In May, a Tor user reported that Cyberoam, a firewall company, had released an update that enabled detection and blocking of meek, among other Tor pluggable transports [109]. Through experiments we determined that the firewall was detecting meek whenever it saw a combination of two features: a specific client TLS fingerprint, and an SNI containing any of our three front domains: www.google.com, a0.awsstatic.com, or ajax.aspnetcdn.com [69]. We verified that changing either the TLS fingerprint or the front domain was sufficient to escape detection. Requiring both features to be present was a clever move by the firewall to limit collateral damage: it did not block those domains for all clients, but only for the subset having a particular TLS fingerprint. I admit that I had not considered the possibility of using TLS and SNI together to make a more precise classifier. We had known since the beginning of the possibility of TLS fingerprinting, which is why we took the trouble to implement browser-based TLS camouflage. The camouflage was performing as intended: even an ordinary Firefox 38 (the basis of Tor Browser, and what meek camouflaged itself as) would be blocked by the firewall when accessing one of the three listed domains. However, Firefox 38 was by that time a year old. I found a source [69] saying that at that time, Firefox 38 made up only 0.38% of desktop browsers, compared to 10.69% for the then-latest Firefox 45 My guess is that the firewall makers considered the small amount of collateral blocking of genuine Firefox 38 users to be acceptable.

In July I received a report of similar behavior by a FortiGuard firewall [72] from Tor user Kanwaljeet Singh Channey. The situation was virtually the same as in the Cyberoam case: the firewall would block connections having a specific TLS fingerprint and a specific SNI. This time, the TLS fingerprint was that of Firefox 45 (which by then Tor Browser had upgraded to); and the specific SNIs were two, not three, omitting www.google.com. As in the previous case, changing either the TLS fingerprint or the front domain was sufficient to get through the firewall.

For reasons not directly related to domain fronting or meek, I had been interested in the blocking situation in Kazakhstan, ever since Tor Metrics reported a sudden drop in the number of users in that country in June 2016 [88]. (See Section 5.4 for other results from Kazakhstan.) I worked with an anonymous collaborator, who reported that meek was blocked in the country since October 2016 or earlier. According to them, changing the front domain would evade the block, but changing the TLS fingerprint didn't help. I did not independently confirm these reports. Kazakhstan remains the only case of country-level blocking of meek that I am aware of.

Starting in July 2016, there was a months-long increase in the number of meek users reported from Brazil [177]. The estimated count went from around 100 to almost 5,000,

peaking in September 2016 before declining again. During parts of this time, over half of all reported meek users were from Brazil. We never got to the bottom of why there should be so many users reported from Brazil in particular. The explanation may be some kind of anomaly; for instance some third-party software that happened to use meek, or a malware infection like the one that caused the shutdown of meek-google. The count of users from Brazil dropped suddenly, from 1,500 almost to zero, on March 3, 2017, which happened also to be the day that I shut down meek-azure pending a migration to new infrastructure. The Brazil count would remain low until rising again in June 2017.

In September 2016, I began mentoring Katherine Li in writing GAEuploader [122], a program to simplify and automate the process of setting up domain fronting. The program automatically uploads the necessary code to Google App Engine, then outputs a bridge specification ready to be pasted into Tor Browser or Orbot. We hoped also that the code would be useful to other projects, like XX-Net [205], that require users to perform the complicated task of uploading code to App Engine. GAEuploader had beta releases in January [121] and November [123] 2017; however the effect on the number of users has so far not been substantial.

Between October 19 and November 10, 2016, the number of meek users decreased globally by about a third [86]. Initially I suspected a censorship event, but the other details didn't add up: the numbers decreased and later recovered simultaneously across many countries, including ones not known for censorship. Discussion with other developers revealed the likely cause: a botched release of Orbot that left some users unable to use the program [79]. Once a fixed release was available, user numbers recovered. As an side effect of this event, we learned that a majority of meek users were using Orbot rather than Tor Browser.

2017: Long-term support

In January 2017, a grant I had been using to pay meek-azure's bandwidth bills ran out. Lacking the means to keep it running, I announced my intention to shut it down [76]. Shortly thereafter, Team Cymru offered to set up their own instances and pay the CDN fees, and so we made plans to migrate meek-azure to the new setup in the next releases. For cost reasons, though, I still had to shut down the old configuration before the new releases of Tor Browser and Orbot were fully ready. I shut down my configuration on March 3. The next release of Tor Browser was on March 7, and the next release of Orbot was on March 22: so there was a period of days or weeks during which meek-azure was non-functional. It would have been better to allow the two configurations to run concurrently for a time, so that users of the old would be able to transparently upgrade to the new—but for cost reasons it was not possible. Perhaps not coincidentally, the surge of users from Brazil, which had started in July 2016, ceased on March 3, the same day I shut down meek-azure before its migration. Handing over control of the infrastructure was a relief to me. I had managed to make sure the monthly bills got paid, but it took more care and attention than I liked. A negative side effect of the migration was that I stopped writing monthly summaries of costs, because I was no longer receiving bills.

Also in January 2017, I became aware of the firewall company Allot Communications, thanks to my anonymous collaborator in the work Kazakhstan work. Allot's marketing materials advertised support for detection of a wide variety of circumvention protocols,

including Tor pluggable transports, Psiphon, and various VPN services [81]. They claimed detection of "Psiphon CDN (Meek mode)" going back to January 2015, and of "TOR (CDN meek)" going back to April 2015. We did not have any Allot devices to experiment with, and I do not know how (or how well) their detectors worked.

In June 2017, the estimated user count from Brazil began to increase again [177], similarly to how it had between July 2016 and March 2017. Just as before, we did not find an explanation for the increase.

The rest of 2017 was fairly quiet. Starting in October, there were reports from China of the disruption of look-like-nothing transports such as obfs4 and Shadowsocks [80], perhaps related to the National Congress of the Communist Party of China that was then about to take place. The disruption did not affect meek or other systems based on domain fronting; in fact the number of meek users in China roughly doubled during that time.

Chapter 7

Snowflake

Snowflake is a new circumvention system currently under development. It is based on peer-to-peer connections through ephemeral proxies that run in web browsers. Snowflake proxies are lightweight: activating one is as easy as browsing to a web page and shutting one down only requires closing the browser tab. They serve only as temporary stepping stones to a full-fledged proxy. Snowflake derives its blocking resistance from having a large number of proxies. A client may use a particular proxy for only seconds or minutes before switching to another. If the censor manages to block the IP address of one proxy, there is little harm, because many other temporary proxies are ready to take its place.

Snowflake [98, 173] is the spiritual successor to flash proxy [84], a system that similarly used browser-based proxies, written in JavaScript. Flash proxy, with obfs2 and obfs3, was one of the first three pluggable transports for Tor [68], but since its introduction in 2013 it never had many users [179]. I believe that its lack of adoption was a result mainly of its incompatibility with NAT (network address translation): its use of the TCP-based WebSocket protocol [64] required clients to follow complicated port forwarding instructions [71]. For that reason, flash proxy was deprecated in 2016 [13].

Snowflake keeps the basic idea of in-browser proxies, but replaces WebSocket with WebRTC [5], a suite of protocols for peer-to-peer communications. Importantly, WebRTC uses UDP for communication, and includes facilities for NAT traversal, allowing most clients to use it without manual configuration. WebRTC mandatorily encrypts its channels, which as a side effect obscures any keywords or byte patterns in the tunneled traffic. (Still leaving open the possibility of detecting the use of WebRTC itself—see Section 7.2.)

Aside from flash proxy, the most similar existing design was a former version of uProxy [184] (an upcoming revision will work differently). uProxy required clients to know a confederate outside the censor's network who could run a proxy. The client would connect through the proxy using WebRTC; the proxy would then directly fetch the client's requested URLs. Snowflake centralizes the proxy discovery process, removing the requirement to arrange one's own proxy outside the firewall. Snowflake proxies are merely dumb pipes to a more capable proxy, allowing them to carry traffic other than web traffic, and preventing them from spying on the client's traffic.

The name Snowflake comes from one of WebRTC's subprotocols, ICE (Interactive Connectivity Establishment) [164], and from the temporary proxies, which resemble snowflakes in their impermanence and uniqueness.

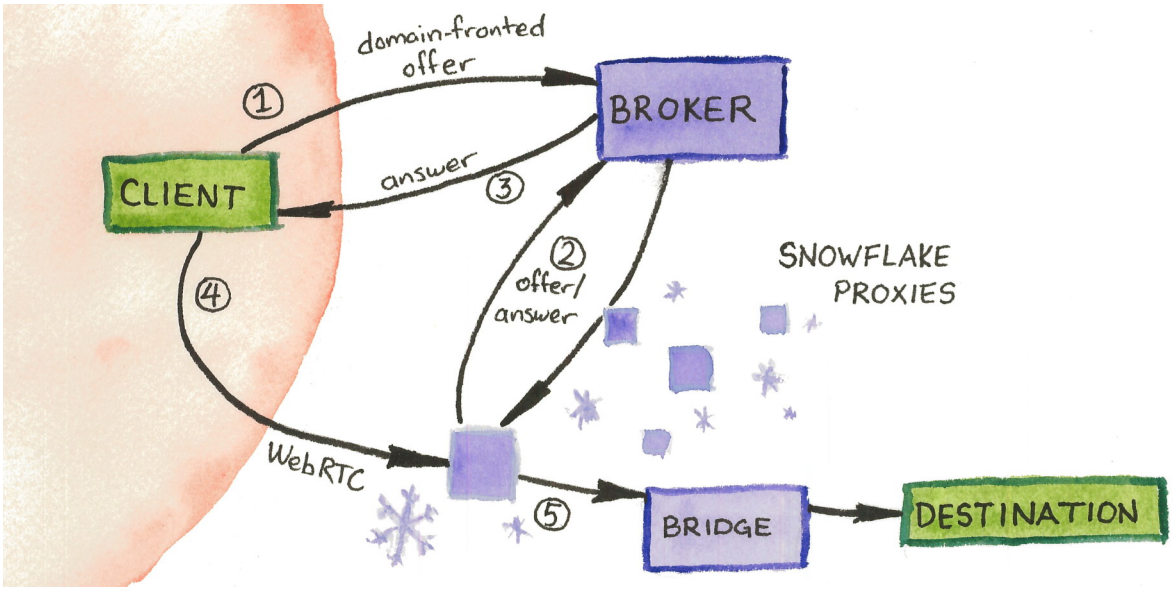

Figure 7.1: Schematic of Snowflake. See Figure 7.2 on page 65 for elaboration on Steps 1, 2, and 3.

Snowflake now exists in an experimental alpha release, incorporated into Tor Browser. My main collaborators on the Snowflake project are Arlo Breault, Mia Gil Epner, Serene Han, and Hooman Mohajeri Moghaddam.

7.1 Design

There are three main components of the Snowflake system. Refer to Figure 7.1.

- many *snowflake proxies*, which communicate with clients over WebRTC and forward their traffic to the bridge
- many *clients*, responsible for initially requesting service and then establishing peer-to-peer connections with snowflake proxies
- a *broker*, an online database that serves to match clients with snowflake proxies
- a *bridge* (so called to distinguish it from the snowflake proxies), a full-featured proxy capable of connecting to any destination

The architecture of the system is influenced by the requirement that proxies run in a browser, and the nature of WebRTC connection establishment, which uses a bidirectional handshake. In our implementation, the bridge is really a Tor bridge. Even though a Tor circuit consists of multiple hops, that fact is abstracted away from the Tor client's perspective; Snowflake does not inherently depend on Tor.

A Snowflake connection happens in multiple steps. In the first phase, called *rendezvous*, the client and snowflake exchange information necessary for a WebRTC connection.

1. The client registers its need for service by sending a message to the broker. The message, called an *offer* [166], contains the client's IP address and other metadata needed to establish a WebRTC connection. How the client sends its offer is further explained below.

2. At some point, a snowflake proxy comes online and polls the broker. The broker hands the client's offer to the snowflake proxy, which sends back its *answer* [166], containing its IP address and other connection metadata the client will need to know.

3. The broker sends back to the client the snowflake's answer message.

At this point rendezvous is finished. The snowflake has the client's offer, and the client has the snowflake's answer, so they have all the information needed to establish a WebRTC connection to each other.

4. The client and snowflake proxy connect to each other using WebRTC.

5. The snowflake proxy connects to the bridge (using WebSocket, though the specific type of channel does not matter for this step).

The snowflake proxy then copies data back and forth between client and bridge until it is terminated. The client's communication with the bridge is encrypted and authenticated end-to-end through the WebRTC tunnel, so the proxy cannot interfere with it. When the snowflake proxy terminates, the client may request a new one. Various optimizations are possible, such as having the client maintain a pool of proxies in order to bridge gaps in connectivity, but we have not implemented and tested them sufficiently to state their effects.

The rendezvous phase bears further explanation. Steps 1, 2, and 3 actually happen synchronously, using interleaved HTTP requests and responses: see Figure 7.2. The client's single request uses domain fronting, but the requests of the snowflake proxies are direct. In Step 1, the client sends a request containing its offer. The broker holds the connection open but does not immediately respond. In Step 2, a snowflake proxy makes a polling request ("do you have any clients for me?") and the broker responds with the client's offer. The snowflake composes its answer and sends it back to the broker in a second HTTP request (linked to the first by a random token). In Step 3, the broker finally responds to the client's initial request by passing on the snowflake proxy's answer. From the client's point of view, it has sent a single request (containing an offer) and received a single response (containing an answer). If no proxy arrives within a time threshold of the client sending its offer, the broker replies with an error message instead. We learned from the experience of running flash proxy that it is not difficult to achieve a proxy arrival rate of several per second, so timeouts ought to be exceptional.

One may ask, if the domain-fronted rendezvous channel is bidirectional and already assumed to be difficult to block, doesn't it suffice for circumvention on its own? The answer is that it does suffice—that's the idea behind meek (Section 6.3). The disadvantage of building a system exclusively on domain fronting, though, is high monetary cost (see Table 6.5 on page 52). Snowflake offloads the bulk of data transfer onto WebRTC, and uses expensive domain fronting only for rendezvous.

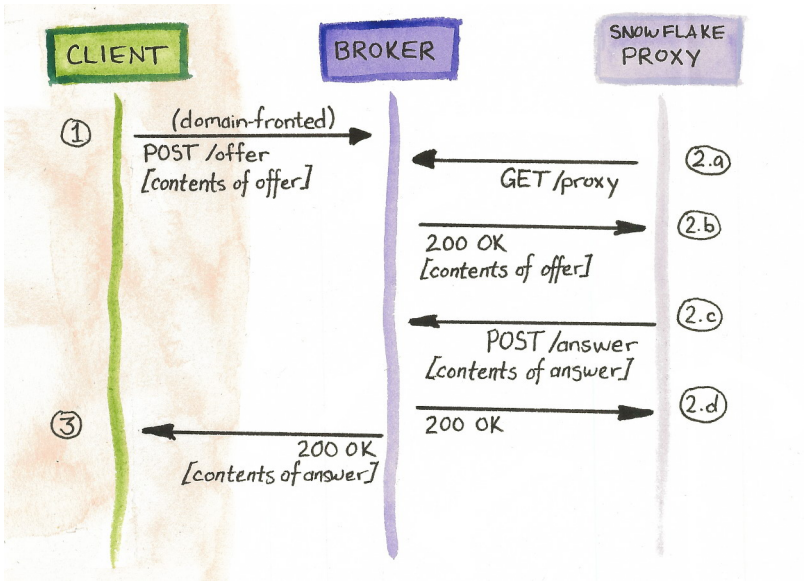

Figure 7.2: Snowflake rendezvous. The client makes only one HTTP request–response pair. In between the client's request and response, the snowflake proxy makes two of its own request–response pairs, the first to learn the client's offer and the second to send back its answer.

There are two reasons why the snowflake proxies forward client traffic to a separate bridge, rather than connecting directly to the client's desired destination. The first is generality: a browser-based proxy can only do the things a browser can do; it can fetch web pages but cannot, for example, open sockets to arbitrary destinations. The second is privacy: the proxies are operated by untrusted, potentially malicious strangers. If they were to exit client traffic directly, they would be able to tamper with it. Furthermore, a malicious *client* could cause a well-meaning proxy to connect to suspicious destinations, potentially getting its operator in trouble. This "many proxies, one bridge" model is essentially untrusted messenger delivery [63], proposed by Feamster et al. in 2003.

WebRTC offers two features that are necessary for Snowflake: 1. it is supported in web browsers, and 2. it deals with NAT. In other respects, though, WebRTC is a nuisance. Its close coupling with browser code makes it difficult to use as a library outside of a browser; a big part of the Snowflake project was to extract the code into a reusable library, go-webrtc [22]. WebRTC comes with a lot of baggage around audio and video codecs, which is useful for some of its intended use cases, but which we would prefer not to have to deal with. Working within a browser environment limits our flexibility, because we cannot access the network directly, but only at arm's length through some API. This has implications for detection by content, as discussed in the next section.

7.2 WebRTC fingerprinting

Snowflake primarily tackles the problem of detection by address. The pool of temporary proxies changes too quickly for a censor to keep up with—or at least that's the idea. Equally important, though, is the problem of detection by content. If Snowflake's protocol has an

easily detectable "tell," then it could be blocked despite its address diversity. Just as with meek we were concerned about TLS fingerprinting (Section 6.2), with Snowflake we are concerned with WebRTC fingerprinting.

Snowflake will always look like WebRTC—that's unavoidable without a major change in architecture. Therefore the best we can hope for is to make Snowflake's WebRTC hard to distinguish from other applications of WebRTC. And that alone is not enough—it also must be that the censor is reluctant to block those other uses of WebRTC.

Mia Gil Epner and I began an investigation into the potential fingerprintability of WebRTC [20, 83]. While preliminary, we were able to find many potential fingerprinting features, and a small survey of applications revealed a diversity of implementation choices in practice.

WebRTC is a stack of interrelated protocols, and leaves implementers much freedom to combined them in different ways. We checked the various protocols in order to find places where implementation choices could facilitate fingerprinting.

Signaling Signaling is WebRTC's term for the exchange of metadata and control data necessary to establish the peer-to-peer connection. WebRTC offers no standard way to do signaling [5 §3]; it is left up to implementers. For example, some implementations do signaling via XMPP, an instant messaging protocol. Snowflake does signaling through the broker, during the rendezvous phase.

ICE ICE (Interactive Connectivity Establishment) [164] is a combination of two protocols. STUN (Session Traversal Utilities for NAT) [165] helps hosts open and maintain a binding in a NAT table. TURN (Traversal Using Relays around NAT) [127] is a way of proxying through a third party when the end hosts' NAT configurations are such that they cannot communicate directly. In STUN, both client and server messages have a number of optional attributes, including one called SOFTWARE that directly specifies the implementation. Furthermore, the very choice of which STUN and TURN servers to use is a choice made by the client.

Media and data channels WebRTC offers media channels (used for audio and video) as well as two kinds of data channels (stream-oriented reliable and datagram-oriented unreliable). All channels are encrypted, however they are encrypted differently according to their type. Media channels use SRTP (Secure Real-time Transport Protocol) [16] and data channels use DTLS (Datagram TLS) [161]. Even though the contents of both are encrypted, an observer can easily distinguish a media channel from a data channel. Applications that use media channels have options for doing key exchange: some borrow the DTLS handshake in a process called DTLS-SRTP [135] and some use SRTP with Security Descriptions (SDES) [11]. Snowflake uses reliable data channels.

DTLS DTLS, as with TLS, offers a wealth of fingerprintable features. Some of the most salient are the protocol version, extensions, the client's offered ciphersuites, and values in the server's certificate.

Snowflake uses a WebRTC library extracted from the Chromium web browser, which mitigates some potential dead-parrot distinguishers [103]. But WebRTC remains complicated and its behavior on the network depends on more than just what library is in use.

We conducted a survey of some WebRTC-using applications in order to get an idea of the implementation choices being made in practice. We tested three applications that use media channels, all chat services: Google Hangouts (https://hangouts.google.com), Facebook Messenger (https://www.messenger.com), and OpenTokRTC (https://opentokrtc.com/). We also tested two applications that use data channels: Snowflake itself and Sharefest (https://github.com/Peer5/ShareFest), a now-defunct file sharing service. Naturally, the network fingerprints of all five applications were distinguishable at some level. Snowflake, by default, uses a Google-operated STUN server, which may be a good choice because so do Hangouts and Sharefest. All applications other than Hangouts used DTLS for key exchange. While the client portions differed, the server certificate was more promising, in all cases having a Common Name of "WebRTC" and a validity of 30 days.

Finally, we wrote a script [82] to detect and fingerprint DTLS handshakes. Running the script on a day's worth of traffic from Lawrence Berkeley National Laboratory turned up only seven handshakes, having three distinct fingerprints. While it is difficult to generalize from one measurement at one site, these results suggest that WebRTC use—at least the forms that use DTLS—is not common. We guessed that Google Hangouts would be the main source of WebRTC connections; however our script would not have found Hangouts connections because Hangouts does not use DTLS.

Bibliography

I strive to provide a URL for each reference whenever possible. On December 15, 2017, I archived each URL at the Internet Archive; or, when that didn't work, at archive.is. If a link is broken, look for an archived version at https://web.archive.org/ or https://archive.is/. Many of the references are also cached in CensorBib, https://censorbib.nymity.ch/.

[1] Nicholas Aase, Jedidiah R. Crandall, Álvaro Díaz, Jeffrey Knockel, Jorge Ocaña Molinero, Jared Saia, Dan Wallach, and Tao Zhu. "Whiskey, Weed, and Wukan on the World Wide Web: On Measuring Censors' Resources and Motivations". In: *Free and Open Communications on the Internet*. USENIX, 2012. https://www.usenix.org/system/files/conference/foci12/foci12-final17.pdf (cit. on p. 34).

[2] Giuseppe Aceto, Alessio Botta, Antonio Pescapè, M. Faheem Awan, Tahir Ahmad, and Saad Qaisar. "Analyzing Internet Censorship in Pakistan". In: *Research and Technologies for Society and Industry*. IEEE, 2016. http://wpage.unina.it/giuseppe.aceto/pub/aceto2016analyzing.pdf (cit. on p. 21).

[3] Giuseppe Aceto, Alessio Botta, Antonio Pescapè, Nick Feamster, M. Faheem Awan, Tahir Ahmad, and Saad Qaisar. "Monitoring Internet Censorship with UBICA". In: *Traffic Monitoring and Analysis*. Springer, 2015. http://wpage.unina.it/giuseppe.aceto/pub/aceto2015monitoring_TMA.pdf (cit. on p. 22).

[4] Percy Alpha. *Google disrupted prior to Tiananmen Anniversary; Mirror sites enable uncensored access to information*. June 2014. https://en.greatfire.org/blog/2014/jun/google-disrupted-prior-tiananmen-anniversary-mirror-sites-enable-uncensored-access (cit. on p. 55).

[5] Harald Alvestrand. *Overview: Real Time Protocols for Browser-based Applications*. IETF, Nov. 2017. https://tools.ietf.org/html/draft-ietf-rtcweb-overview-19 (cit. on pp. 62, 66).

[6] Collin Anderson. *Dimming the Internet: Detecting Throttling as a Mechanism of Censorship in Iran*. Tech. rep. University of Pennsylvania, 2013. https://arxiv.org/abs/1306.4361v1 (cit. on p. 20).

[7] Collin Anderson, Roger Dingledine, Nima Fatemi, harmony, and mttp. *Vanilla Tor Connectivity Issues In Iran -- Directory Authorities Blocked?* July 2014. https://bugs.torproject.org/12727 (cit. on p. 45).

[8] Collin Anderson, Philipp Winter, and Roya. "Global Network Interference Detection over the RIPE Atlas Network". In: *Free and Open Communications on the Internet*. USENIX, 2014. https://www.usenix.org/system/files/conference/foci14/foci14-anderson.pdf (cit. on p. 22).

[9] Daniel Anderson. "Splinternet Behind the Great Firewall of China". In: *ACM Queue* 10.11 (2012), p. 40. https://queue.acm.org/detail.cfm?id=2405036 (cit. on pp. 15, 20).

[10] Ross J. Anderson. "The Eternity Service". In: *Theory and Applications of Cryptology*. CTU Publishing House, 1996, pp. 242–253. https://www.cl.cam.ac.uk/~rja14/Papers/eternity.pdf (cit. on p. 3).

[11] Flemming Andreasen, Mark Baugher, and Dan Wing. *Session Description Protocol (SDP) Security Descriptions for Media Streams*. IETF, July 2006. https://tools.ietf.org/html/rfc4568 (cit. on p. 66).

[12] Anonymous. "Towards a Comprehensive Picture of the Great Firewall's DNS Censorship". In: *Free and Open Communications on the Internet*. USENIX, 2014. https://www.usenix.org/system/files/conference/foci14/foci14-anonymous.pdf (cit. on p. 19).

[13] Anonymous, David Fifield, Georg Koppen, Mark Smith, and Yawning Angel. *Remove Flashproxy from Tor Browser*. Oct. 2015. https://bugs.torproject.org/17428 (cit. on p. 62).

[14] Simurgh Aryan, Homa Aryan, and J. Alex Halderman. "Internet Censorship in Iran: A First Look". In: *Free and Open Communications on the Internet*. USENIX, 2013. https://censorbib.nymity.ch/pdf/Aryan2013a.pdf (cit. on pp. 20, 45).

[15] Geremie R. Barme and Ye Sang. "The Great Firewall of China". In: *Wired* (June 1997). https://archive.wired.com/wired/archive/5.06/china_pr.html (cit. on p. 15).

[16] Mark Baugher, David McGrew, Mats Naslund, Elisabetta Carrara, and Karl Norrman. *The Secure Real-time Transport Protocol (SRTP)*. IETF, Mar. 2004. https://tools.ietf.org/html/rfc3711 (cit. on p. 66).

[17] Bryce Boe. *Bypassing Gogo's Inflight Internet Authentication*. Mar. 2012. http://bryceboe.com/2012/03/12/bypassing-gogos-inflight-internet-authentication/ (cit. on p. 49).

[18] David Borman, Bob Braden, Van Jacobson, and Richard Scheffenegger. *TCP Extensions for High Performance*. IETF, Sept. 2014. https://tools.ietf.org/html/rfc7323 (cit. on p. 31).

[19] BreakWa11. *ShadowSocks*协议的弱点分析和改进. Aug. 2015. https://web.archive.org/web/20160829052958/https://github.com/breakwa11/shadowsocks-rss/issues/38 (cit. on pp. 26, 28).

[20] Arlo Breault, David Fifield, and Mia Gil Epner. *Snowflake/Fingerprinting*. Tor Bug Tracker & Wiki. https://trac.torproject.org/projects/tor/wiki/doc/Snowflake/Fingerprinting (cit. on p. 66).

[21] Arlo Breault, David Fifield, and George Kadianakis. *Registration over App Engine*. May 2013. https://bugs.torproject.org/8860 (cit. on p. 53).

[22] Arlo Breault and Serene Han. *go-webrtc*. https://github.com/keroserene/go-webrtc (cit. on p. 65).

[23] Chad Brubaker, Amir Houmansadr, and Vitaly Shmatikov. "CloudTransport: Using Cloud Storage for Censorship-Resistant Networking". In: *Privacy Enhancing Technologies Symposium*. Springer, 2014. https://petsymposium.org/2014/papers/paper_68.pdf (cit. on pp. 8, 10, 49).

[24] Willow Brugh. *San Francisco Hackathon/DiscoTech (+ RightsCon + Responsible Data Forum)*. Mar. 2014. http://codesign.mit.edu/2014/03/sfdiscotech/ (cit. on p. 55).

[25] Sam Burnett, Nick Feamster, and Santosh Vempala. "Chipping Away at Censorship Firewalls with User-Generated Content". In: *USENIX Security Symposium*. USENIX, 2010. https://www.usenix.org/event/sec10/tech/full_papers/Burnett.pdf (cit. on p. 8).

[26] Cormac Callanan, Hein Dries-Ziekenheiner, Alberto Escudero-Pascual, and Robert Guerra. *Leaping Over the Firewall: A Review of Censorship Circumvention Tools*. Tech. rep. Freedom House, 2011. https://freedomhouse.org/report/special-reports/leaping-over-firewall-review-censorship-circumvention-tools (cit. on p. 23).

[27] Abdelberi Chaabane, Terence Chen, Mathieu Cunche, Emiliano De Cristofaro, Arik Friedman, and Mohamed Ali Kaafar. "Censorship in the Wild: Analyzing Internet Filtering in Syria". In: *Internet Measurement Conference*. ACM, 2014. http://conferences2.sigcomm.org/imc/2014/papers/p285.pdf (cit. on p. 21).

[28] The Citizen Lab. *Psiphon*. Oct. 2006. https://web.archive.org/web/20061026081356/http://psiphon.civisec.org/ (cit. on p. 15).

[29] Erinn Clark. *Tor Browser 3.6.4 and 4.0-alpha-1 are released*. The Tor Blog. Aug. 2014. https://blog.torproject.org/tor-browser-364-and-40-alpha-1-are-released (cit. on pp. 11, 55).

[30] Richard Clayton. "Failures in a Hybrid Content Blocking System". In: *Privacy Enhancing Technologies*. Springer, 2006, pp. 78–92. https://www.cl.cam.ac.uk/~rnc1/cleanfeed.pdf (cit. on p. 18).

[31] Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson. "Ignoring the Great Firewall of China". In: *Privacy Enhancing Technologies*. Springer, 2006, pp. 20–35. https://www.cl.cam.ac.uk/~rnc1/ignoring.pdf (cit. on pp. 14, 19, 43).

[32] Jedidiah R. Crandall, Masashi Crete-Nishihata, and Jeffrey Knockel. "Forgive Us our SYNs: Technical and Ethical Considerations for Measuring Internet Filtering". In: *Ethics in Networked Systems Research*. ACM, 2015. https://censorbib.nymity.ch/pdf/Crandall2015a.pdf (cit. on p. 18).

[33] Jedidiah R. Crandall, Daniel Zinn, Michael Byrd, Earl Barr, and Rich East. "ConceptDoppler: A Weather Tracker for Internet Censorship". In: *Computer and Communications Security*. ACM, 2007, pp. 352–365. http://www.csd.uoc.gr/~hy558/papers/conceptdoppler.pdf (cit. on pp. 15, 19).

[34] Creative Commons. *CC0 1.0 Universal*. https://creativecommons.org/publicdomain/zero/1.0/ (cit. on p. 55).

[35] Elena Cresci. "How to get around Turkey's Twitter ban". In: *The Guardian* (Mar. 2014). https://www.theguardian.com/world/2014/mar/21/how-to-get-around-turkeys-twitter-ban (cit. on p. 16).

[36] Eric Cronin, Micah Sherr, and Matt Blaze. *The Eavesdropper's Dilemma*. Tech. rep. MS-CIS-05-24. Department of Computer and Information Science, University of Pennsylvania, 2005. http://www.crypto.com/papers/internet-tap.pdf (cit. on p. 14).

[37] Alberto Dainotti, Claudio Squarcella, Emile Aben, Kimberly C. Claffy, Marco Chiesa, Michele Russo, and Antonio Pescapè. "Analysis of Country-wide Internet Outages Caused by Censorship". In: *Internet Measurement Conference*. ACM, 2011, pp. 1–18. http://conferences.sigcomm.org/imc/2011/docs/p1.pdf (cit. on p. 20).

[38] Jakub Dalek, Bennett Haselton, Helmi Noman, Adam Senft, Masashi Crete-Nishihata, Phillipa Gill, and Ronald J. Deibert. "A Method for Identifying and Confirming the Use of URL Filtering Products for Censorship". In: *Internet Measurement Conference*. ACM, 2013. http://conferences.sigcomm.org/imc/2013/papers/imc112s-dalekA.pdf (cit. on p. 21).

[39] Jakub Dalek, Adam Senft, Masashi Crete-Nishihata, and Ron Deibert. *O Pakistan, We Stand on Guard for Thee: An Analysis of Canada-based Netsweeper's Role in Pakistan's Censorship Regime*. June 2013. https://citizenlab.ca/2013/06/o-pakistan/ (cit. on p. 21).

[40] Deloitte. *The economic impact of disruptions to Internet connectivity*. Oct. 2016. https://globalnetworkinitiative.org/sites/default/files/The-Economic-Impact-of-Disruptions-to-Internet-Connectivity-Deloitte.pdf (cit. on p. 9).

[41] denverroot, Roger Dingledine, Aaron Gibson, hrimfaxi, George Kadianakis, Andrew Lewman, OlgieD, Mike Perry, Fabio Pietrosanti, and quick-dudley. *Bridge easily detected by GFW*. Oct. 2011. https://bugs.torproject.org/4185 (cit. on pp. 26, 27).

[42] Tim Dierks and Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. IETF, Aug. 2008. https://tools.ietf.org/html/rfc5246 (cit. on p. 47).

[43] Roger Dingledine. *Obfsproxy: the next step in the censorship arms race*. The Tor Blog. Feb. 2012. https://blog.torproject.org/obfsproxy-next-step-censorship-arms-race (cit. on pp. 10, 26, 27, 34).

[44] Roger Dingledine. *Please run a bridge relay! (was Re: Tor 0.2.0.13-alpha is out)*. tor-talk mailing list. Dec. 2007. https://lists.torproject.org/pipermail/tor-talk/2007-December/003854.html (cit. on p. 12).

[45] Roger Dingledine. *Strategies for getting more bridge addresses*. Tech. rep. 2011-05-001. The Tor Project, May 2011. https://research.torproject.org/techreports/strategies-getting-more-bridge-addresses-2011-05-13.pdf (cit. on p. 12).

[46] Roger Dingledine. *Ten ways to discover Tor bridges*. Tech. rep. 2011-10-002. The Tor Project, Oct. 2011. https://research.torproject.org/techreports/ten-ways-discover-tor-bridges-2011-10-31.pdf (cit. on pp. 12, 24).

[47] Roger Dingledine, David Fifield, George Kadianakis, Lunar, Runa Sandvik, and Philipp Winter. *GFW actively probes obfs2 bridges*. Mar. 2013. https://bugs.torproject.org/8591 (cit. on pp. 26, 27).

[48] Roger Dingledine, Arturo Filastò, George Kadianakis, Nick Mathewson, and Philipp Winter. *GFW probes based on Tor's SSL cipher list*. Dec. 2011. https://bugs.torproject.org/4744 (cit. on pp. 26, 27, 51).

[49] Roger Dingledine and Nick Mathewson. *Design of a blocking-resistant anonymity system*. Tech. rep. 2006-11-001. The Tor Project, Nov. 2006. https://research.torproject.org/techreports/blocking-2006-11.pdf (cit. on pp. 11, 12, 15, 24, 33).

[50] Roger Dingledine and Nick Mathewson. *Tor Protocol Specification*. Sept. 2017. https://spec.torproject.org/tor-spec (cit. on p. 32).

[51] Bill Dong. *A report about national DNS spoofing in China on Sept. 28th*. Oct. 2002. https://web.archive.org/web/20021015121616/http://www.dit-inc.us/hj-09-02.html (cit. on p. 18).

[52] Maximillian Dornseif. "Government mandated blocking of foreign Web content". In: *DFN-Arbeitstagung über Kommunikationsnetze*. Gesellschaft für Informatik, 2003, pp. 617–647. https://censorbib.nymity.ch/pdf/Dornseif2003a.pdf (cit. on p. 18).

[53] Eva Dou and Alistair Barr. *U.S. Cloud Providers Face Backlash From China's Censors*. *The Wall Street Journal*. Mar. 2015. https://www.wsj.com/articles/u-s-cloud-providers-face-backlash-from-chinas-censors-1426541126 (cit. on p. 56).

[54] Frederick Douglas, Rorshach, Weiyang Pan, and Matthew Caesar. "Salmon: Robust Proxy Distribution for Censorship Circumvention". In: *Privacy Enhancing Technologies* 2016.4 (2016), pp. 4–20. https://www.degruyter.com/downloadpdf/j/popets.2016.2016.issue-4/popets-2016-0026/popets-2016-0026.xml (cit. on p. 12).

[55] Matthew Dunwoody. *APT29 Domain Fronting With TOR*. FireEye Threat Research Blog. Mar. 2017. https://www.fireeye.com/blog/threat-research/2017/03/apt29_domain_frontin.html (cit. on p. 59).

[56] Matthew Dunwoody and Nick Carr. *No Easy Breach*. DerbyCon. Sept. 2016. https://www.slideshare.net/MatthewDunwoody1/no-easy-breach-derby-con-2016 (cit. on p. 59).

[57] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. "ZMap: Fast Internet-Wide Scanning and its Security Applications". In: *USENIX Security Symposium*. USENIX, 2013. https://zmap.io/paper.pdf (cit. on pp. 12, 24).

[58] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. "Protocol Misidentification Made Easy with Format-Transforming Encryption". In: *Computer and Communications Security*. ACM, 2013. https://eprint.iacr.org/2012/494.pdf (cit. on p. 10).

[59] Don Eastlake. *Transport Layer Security (TLS) Extensions: Extension Definitions*. IETF, Jan. 2011. https://tools.ietf.org/html/rfc6066 (cit. on p. 47).

[60] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. "Examining How the Great Firewall Discovers Hidden Circumvention Servers". In: *Internet Measurement Conference*. ACM, 2015. http://conferences2.sigcomm.org/imc/2015/papers/p445.pdf (cit. on pp. 20, 26–28, 30, 43).

[61] Roya Ensafi, Philipp Winter, Abdullah Mueen, and Jedidiah R. Crandall. "Analyzing the Great Firewall of China Over Space and Time". In: *Privacy Enhancing Technologies* 2015.1 (2015). https://censorbib.nymity.ch/pdf/Ensafi2015a.pdf (cit. on pp. 21, 22).

[62] Nick Feamster, Magdalena Balazinska, Greg Harfst, Hari Balakrishnan, and David Karger. "Infranet: Circumventing Web Censorship and Surveillance". In: *USENIX Security Symposium*. USENIX, 2002. http://wind.lcs.mit.edu/papers/usenixsec2002.pdf (cit. on pp. 10, 16).

[63] Nick Feamster, Magdalena Balazinska, Winston Wang, Hari Balakrishnan, and David Karger. "Thwarting Web Censorship with Untrusted Messenger Discovery". In: *Privacy Enhancing Technologies*. Springer, 2003, pp. 125–140. http://nms.csail.mit.edu/papers/disc-pet2003.pdf (cit. on p. 65).

[64] Ian Fette and Alexey Melnikov. *The WebSocket Protocol*. IETF, Dec. 2011. https://tools.ietf.org/html/rfc6455 (cit. on p. 62).

[65] Roy Fielding and Julian Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. IETF, June 2014. https://tools.ietf.org/html/rfc7230 (cit. on p. 47).

[66] David Fifield. *A simple HTTP transport and big ideas*. tor-dev mailing list. Jan. 2014. https://lists.torproject.org/pipermail/tor-dev/2014-January/006159.html (cit. on p. 54).

[67] David Fifield. *Big performance improvement for meek-azure*. tor-dev mailing list. Apr. 2015. https://lists.torproject.org/pipermail/tor-dev/2015-April/008637.html (cit. on p. 56).

[68] David Fifield. *Combined flash proxy + pyobfsproxy browser bundles*. The Tor Blog. Jan. 2013. https://blog.torproject.org/combined-flash-proxy-pyobfsproxy-browser-bundles (cit. on pp. 26, 27, 62).

[69] David Fifield. *Cyberoam firewall blocks meek by TLS signature*. Network Traffic Obfuscation mailing list. May 2016. https://groups.google.com/d/topic/traffic-obf/BpFSCVgi5rs (cit. on p. 59).

[70] David Fifield. *Estimating censorship lag by obfs4 blocking*. tor-dev mailing list. Feb. 2015. https://lists.torproject.org/pipermail/tor-dev/2015-February/008222.html (cit. on p. 34).

[71] David Fifield. *Flash proxy howto*. Tor Bug Tracker & Wiki. Mar. 2014. https://trac.torproject.org/projects/tor/wiki/doc/PluggableTransports/FlashProxy/Howto (cit. on p. 62).

[72] David Fifield. *FortiGuard firewall blocks meek by TLS signature*. Network Traffic Obfuscation mailing list. July 2016. https://groups.google.com/d/topic/traffic-obf/fwAN-WWz2Bk (cit. on p. 59).

[73] David Fifield. *GoAgent: Further notes on App Engine and speculation about a pluggable transport*. Tor Bug Tracker & Wiki. Oct. 2013. https://trac.torproject.org/projects/tor/wiki/doc/GoAgent?action=diff&version=2&old_version=1 (cit. on p. 53).

[74] David Fifield. *How to use the "meek" pluggable transport*. The Tor Blog. Aug. 2015. https://blog.torproject.org/how-use-meek-pluggable-transport (cit. on p. 55).

[75] David Fifield. *HOWTO use Lantern as a pluggable transport*. tor-dev mailing list. Mar. 2014. https://lists.torproject.org/pipermail/tor-dev/2014-March/006356.html (cit. on p. 55).

[76] David Fifield. *meek-azure funding has run out*. tor-dev mailing list. Jan. 2017. https://lists.torproject.org/pipermail/tor-project/2017-January/000881.html (cit. on p. 60).

[77] David Fifield. *Outage of meek-azure*. tor-dev mailing list. Aug. 2015. https://lists.torproject.org/pipermail/tor-talk/2015-August/038780.html (cit. on p. 57).

[78] David Fifield. *Why the seeming correlation between flash proxy and meek on metrics graphs?* tor-dev mailing list. Sept. 2014. https://lists.torproject.org/pipermail/tor-dev/2014-September/007484.html (cit. on p. 55).

[79] David Fifield, Adam Fisk, Nathan Freitas, and Percy Wegmann. *meek seems blocked in China since 2016-10-19*. Network Traffic Obfuscation mailing list. Oct. 2016. https://groups.google.com/d/topic/traffic-obf/CSJLt3t-_OI (cit. on p. 60).

[80] David Fifield, Vinicius Fortuna, Sergey Frolov, b.l. masters, Will Scott, Tom (hexuxin), and Brandon Wiley. *Reports of China disrupting shadowsocks*. Oct. 2017. https://groups.google.com/d/msg/traffic-obf/dqw6CQLR944/StgigdK0BAAJ (cit. on p. 61).

[81] David Fifield, Vinicius Fortuna, Philipp Winter, and Eric Wustrow. *Allot Communications*. Network Traffic Obfuscation mailing list. Jan. 2017. https://groups.google.com/d/topic/traffic-obf/yzxlLpFyXLI (cit. on p. 61).

[82] David Fifield and Mia Gil Epner. *DTLS-fingerprint*. May 2016. https://github.com/miagilepner/DTLS-fingerprint/ (cit. on p. 67).

[83] David Fifield and Mia Gil Epner. *Fingerprintability of WebRTC*. Tech. rep. May 2016. https://arxiv.org/pdf/1605.08805v1.pdf (cit. on p. 66).

[84] David Fifield, Nate Hardison, Jonathan Ellithorpe, Emily Stark, Roger Dingledine, Phillip Porras, and Dan Boneh. "Evading Censorship with Browser-Based Proxies". In: *Privacy Enhancing Technologies Symposium*. Springer, 2012, pp. 239–258. https://www.bamsoftware.com/papers/flashproxy.pdf (cit. on pp. 13, 53, 62).

[85] David Fifield, George Kadianakis, Georg Koppen, and Mark Smith. *Make bundles featuring meek*. Feb. 2014. https://bugs.torproject.org/10935 (cit. on p. 54).

[86] David Fifield and Georg Koppen. *Unexplained drop in meek users, 2016-10-19 to 2016-11-10*. Oct. 2016. https://bugs.torproject.org/20495 (cit. on p. 60).

[87] David Fifield, Georg Koppen, and Klaus Layer. *Update the meek-amazon fingerprint to B9E7141C594AF25699E0079C1F0146F409495296*. Oct. 2015. https://bugs.torproject.org/17473 (cit. on p. 57).

[88] David Fifield and kzblocked. *Kazakhstan 2016–2017*. OONI Censorship Wiki. June 2017. https://trac.torproject.org/projects/tor/wiki/doc/OONI/censorshipwiki/CensorshipByCountry/Kazakhstan#a20348 (cit. on pp. 45, 59).

[89] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. "Blocking-resistant communication through domain fronting". In: *Privacy Enhancing Technologies* 2015.2 (2015). https://www.bamsoftware.com/papers/fronting/ (cit. on pp. 13, 50, 52, 53, 56).

[90] David Fifield, Karsten Loesing, Isis Agora Lovecruft, and Yawning Angel. *meek's reflector should forward the client's IP address/port to the bridge*. Sept. 2014. https://bugs.torproject.org/13171 (cit. on p. 58).

[91] David Fifield and Lynn Tsai. "Censors' Delay in Blocking Circumvention Proxies". In: *Free and Open Communications on the Internet*. USENIX, 2016. https://www.usenix.org/conference/foci16/workshop-program/presentation/fifield (cit. on p. 33).

[92] Arturo Filastò and Jacob Appelbaum. "OONI: Open Observatory of Network Interference". In: *Free and Open Communications on the Internet*. USENIX, 2012. https://www.usenix.org/system/files/conference/foci12/foci12-final12.pdf (cit. on p. 22).

[93] Nathan Freitas. *Orbot v15-alpha-3 with VPN and Meek!* guardian-dev mailing list. Feb. 2015. https://lists.mayfirst.org/pipermail/guardian-dev/2015-February/004243.html (cit. on p. 56).

[94] Sergey Frolov, Fred Douglas, Will Scott, Allison McDonald, Benjamin VanderSloot, Rod Hynes, Adam Kruger, Michalis Kallitsis, David G. Robinson, Steve Schultze, Nikita Borisov, Alex Halderman, and Eric Wustrow. "An ISP-Scale Deployment of TapDance". In: *Free and Open Communications on the Internet*. USENIX, 2017. https://www.usenix.org/system/files/conference/foci17/foci17-paper-frolov_0.pdf (cit. on p. 49).

[95] John Geddes, Max Schuchard, and Nicholas Hopper. "Cover Your ACKs: Pitfalls of Covert Channel Censorship Circumvention". In: *Computer and Communications Security*. ACM, 2013. https://www-users.cs.umn.edu/~hopper/ccs13-cya.pdf (cit. on p. 9).

[96] Google. *China, All Products, May 31, 2014–Present*. Google Transparency Report. July 2014. https://www.google.com/transparencyreport/traffic/disruptions/124/ (cit. on p. 55).

[97] Google Cloud Platform. *Service Specific Terms*. Mar. 2015. https://web.archive.org/web/20150326000133/https://cloud.google.com/terms/service-terms (cit. on p. 57).

[98] Serene Han. *Snowflake Technical Overview*. Jan. 2017. https://keroserene.net/snowflake/technical/ (cit. on p. 62).

[99] Bennett Haselton. *Circumventor*. Peacefire. http://peacefire.org/circumventor/ (cit. on p. 15).

[100] Bennett Haselton. *Peacefire Censorware Pages*. Peacefire. http://www.peacefire.org/censorware/ (cit. on p. 15).

[101] Huifeng He. *Google breaks through China's Great Firewall ... but only for just over an hour*. South China Morning Post. Mar. 2016. http://www.scmp.com/tech/china-tech/article/1931301/google-breaks-through-chinas-great-firewall-only-just-over-hour (cit. on p. 40).

[102] hellofwy, Max Lv, Mygod, Rio, and Siyuan Ren. *SIP007 - Per-session subkey*. Jan. 2017. https://github.com/shadowsocks/shadowsocks-org/issues/42 (cit. on pp. 26, 28).

[103] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. "The Parrot is Dead: Observing Unobservable Network Communications". In: *Symposium on Security & Privacy*. IEEE, 2013. https://people.cs.umass.edu/~amir/papers/parrot.pdf (cit. on pp. 8, 9, 66).

[104] Amir Houmansadr, Giang T. K. Nguyen, Matthew Caesar, and Nikita Borisov. "Cirripede: Circumvention Infrastructure using Router Redirection with Plausible Deniability". In: *Computer and Communications Security*. ACM, 2011, pp. 187–200. https://hatswitch.org/~nikita/papers/cirripede-ccs11.pdf (cit. on pp. 8, 49).

[105] Amir Houmansadr, Thomas Riedl, Nikita Borisov, and Andrew Singer. "I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention". In: *Network and Distributed System Security*. The Internet Society, 2013. https://people.cs.umass.edu/~amir/papers/FreeWave.pdf (cit. on pp. 8, 10).

[106] Amir Houmansadr, Edmund L. Wong, and Vitaly Shmatikov. "No Direction Home: The True Cost of Routing Around Decoys". In: *Network and Distributed System Security*. The Internet Society, 2014. http://dedis.cs.yale.edu/dissent/papers/nodirection.pdf (cit. on p. 13).

[107] *ICLab*. https://iclab.org/ (cit. on p. 22).

[108] Ben Jones, Roya Ensafi, Nick Feamster, Vern Paxson, and Nick Weaver. "Ethical Concerns for Censorship Measurement". In: *Ethics in Networked Systems Research*. ACM, 2015. https://www.icir.org/vern/papers/censorship-meas.nsethics15.pdf (cit. on p. 18).

[109] Justin. *Pluggable Transports and DPI*. tor-dev mailing list. May 2016. https://lists.torproject.org/pipermail/tor-talk/2016-May/040898.html (cit. on p. 59).

[110] George Kadianakis and Nick Mathewson. *obfs2 (The Twobfuscator)*. Jan. 2011. https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs2/obfs2-protocol-spec.txt (cit. on p. 10).

[111] George Kadianakis and Nick Mathewson. *obfs3 (The Threebfuscator)*. Jan. 2013. https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs3/obfs3-protocol-spec.txt (cit. on p. 10).

[112] Josh Karlin, Daniel Ellard, Alden W. Jackson, Christine E. Jones, Greg Lauer, David P. Mankins, and W. Timothy Strayer. "Decoy Routing: Toward Unblockable Internet Communication". In: *Free and Open Communications on the Internet*. USENIX, 2011. https://www.usenix.org/legacy/events/foci11/tech/final_files/Karlin.pdf (cit. on p. 49).

[113] Sheharbano Khattak, Tariq Elahi, Laurent Simon, Colleen M. Swanson, Steven J. Murdoch, and Ian Goldberg. "SoK: Making Sense of Censorship Resistance Systems". In: *Privacy Enhancing Technologies* 2016.4 (2016), pp. 37–61. https://www.degruyter.com/downloadpdf/j/popets.2016.2016.issue-4/popets-2016-0028/popets-2016-0028.xml (cit. on pp. 6, 8, 10, 14).

[114] Sheharbano Khattak, Mobin Javed, Philip D. Anderson, and Vern Paxson. "Towards Illuminating a Censorship Monitor's Model to Facilitate Evasion". In: *Free and Open Communications on the Internet*. USENIX, 2013. https://censorbib.nymity.ch/pdf/Khattak2013a.pdf (cit. on pp. 15, 20).

[115] Sheharbano Khattak, Mobin Javed, Syed Ali Khayam, Zartash Afzal Uzmi, and Vern Paxson. "A Look at the Consequences of Internet Censorship Through an ISP Lens". In: *Internet Measurement Conference*. ACM, 2014. http://conferences2.sigcomm.org/imc/2014/papers/p271.pdf (cit. on p. 20).

[116] Gary King, Jennifer Pan, and Margaret E. Roberts. "How Censorship in China Allows Government Criticism but Silences Collective Expression". In: *American Political Science Review* (2012). https://gking.harvard.edu/files/censored.pdf (cit. on p. 34).

[117] Jeffrey Knockel, Lotus Ruan, and Masashi Crete-Nishihata. "Measuring Decentralization of Chinese Keyword Censorship via Mobile Games". In: *Free and Open Communications on the Internet*. USENIX, 2017. https://www.usenix.org/system/files/conference/foci17/foci17-paper-knockel.pdf (cit. on p. 34).

[118] Stefan Köpsell and Ulf Hillig. "How to Achieve Blocking Resistance for Existing Systems Enabling Anonymous Web Surfing". In: *Workshop on Privacy in the Electronic Society*. ACM, 2004, pp. 47–58. https://censorbib.nymity.ch/pdf/Koepsell2004a.pdf (cit. on pp. 3, 6, 12, 49).

[119] *Lantern*. https://getlantern.org/ (cit. on p. 50).

[120] Bruce Leidl. *obfuscated-openssh*. 2009. https://github.com/brl/obfuscated-openssh (cit. on p. 10).

[121] Katherine Li. *GAEuploader*. tor-dev mailing list. Jan. 2017. https://lists.torproject.org/pipermail/tor-dev/2017-January/011812.html (cit. on p. 60).

[122] Katherine Li. *GAEuploader*. https://github.com/katherinelitor/GAEuploader (cit. on p. 60).

[123] Katherine Li. *GAEuploader now supports Windows*. tor-dev mailing list. Nov. 2017. https://lists.torproject.org/pipermail/tor-dev/2017-November/012622.html (cit. on p. 60).

[124] Karsten Loesing and Nick Mathewson. *BridgeDB specification*. Dec. 2013. https://spec.torproject.org/bridgedb-spec (cit. on p. 12).

[125] Graham Lowe, Patrick Winters, and Michael L. Marcus. *The Great DNS Wall of China*. Tech. rep. New York University, 2007. https://censorbib.nymity.ch/pdf/Lowe2007a.pdf (cit. on p. 19).

[126] Max Lv and Rio. *AEAD Ciphers*. https://shadowsocks.org/en/spec/AEAD-Ciphers.html (cit. on p. 25).

[127] Rohan Mahy, Philip Matthews, and Jonathan Rosenberg. *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*. IETF, Apr. 2010. https://tools.ietf.org/html/rfc5766 (cit. on p. 66).

[128] Marek Majkowski. *Fun with The Great Firewall*. July 2013. https://idea.popcount.org/2013-07-11-fun-with-the-great-firewall/ (cit. on pp. 26, 27).

[129] Bill Marczak, Nicholas Weaver, Jakub Dalek, Roya Ensafi, David Fifield, Sarah McKune, Arn Rey, John Scott-Railton, Ron Deibert, and Vern Paxson. "An Analysis of China's 'Great Cannon'". In: *Free and Open Communications on the Internet*. USENIX, 2015. https://www.usenix.org/system/files/conference/foci15/foci15-paper-marczak.pdf (cit. on pp. 21, 56).

[130] Morgan Marquis-Boire, Jakub Dalek, and Sarah McKune. *Planet Blue Coat: Mapping Global Censorship and Surveillance Tools*. Jan. 2013. https://citizenlab.ca/2013/01/planet-blue-coat-mapping-global-censorship-and-surveillance-tools/ (cit. on p. 21).

[131] James Marshall. *CGIProxy*. https://jmarshall.com/tools/cgiproxy/ (cit. on p. 15).

[132] David Martin and Andrew Schulman. "Deanonymizing Users of the SafeWeb Anonymizing Service". In: *USENIX Security Symposium*. USENIX, 2002. https://www.usenix.org/legacy/publications/library/proceedings/sec02/martin.html (cit. on p. 15).

[133] Srdjan Matic, Carmela Troncoso, and Juan Caballero. "Dissecting Tor Bridges: a Security Evaluation of Their Private and Public Infrastructures". In: *Network and Distributed System Security*. The Internet Society, 2017. https://software.imdea.org/~juanca/papers/torbridges_ndss17.pdf (cit. on pp. 13, 25, 33).

[134] Damon McCoy, Jose Andre Morales, and Kirill Levchenko. "Proximax: A Measurement Based System for Proxies Dissemination". In: *Financial Cryptography and Data Security*. Springer, 2011. https://cseweb.ucsd.edu/~klevchen/mml-fc11.pdf (cit. on p. 12).

[135] David McGrew and Eric Rescorla. *Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)*. IETF, May 2010. https://tools.ietf.org/html/rfc5764 (cit. on p. 66).

[136] Jon McLachlan and Nicholas Hopper. "On the risks of serving whenever you surf: Vulnerabilities in Tor's blocking resistance design". In: *Workshop on Privacy in the Electronic Society*. ACM, 2009. https://www-users.cs.umn.edu/~hopper/surf_and_serve.pdf (cit. on p. 24).

[137] *meek*. Tor Bug Tracker & Wiki. https://trac.torproject.org/projects/tor/wiki/doc/meek (cit. on pp. 52, 56, 57).

[138] Brock N. Meeks and Declan B. McCullagh. *Jacking in from the "Keys to the Kingdom" Port*. CyberWire Dispatch. July 1996. https://cyberwire.com/cwd/cwd.96.07.03.html (cit. on p. 15).

[139] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. "SkypeMorph: Protocol Obfuscation for Tor Bridges". In: *Computer and Communications Security*. ACM, 2012. https://www.cypherpunks.ca/~iang/pubs/skypemorph-ccs.pdf (cit. on p. 10).

[140] Rich Morin. "The Limits of Control". In: *Unix Review* (June 1996). http://cfcl.com/rdm/Pubs/tin/P/199606.shtml (cit. on p. 15).

[141] Zubair Nabi. "The Anatomy of Web Censorship in Pakistan". In: *Free and Open Communications on the Internet*. USENIX, 2013. https://censorbib.nymity.ch/pdf/Nabi2013a.pdf (cit. on p. 20).

[142] NetFreedom Pioneers. *Toosheh*. https://www.toosheh.org/en.html (cit. on p. 14).

[143] Leif Nixon. *Some observations on the Great Firewall of China*. Nov. 2011. https://www.nsc.liu.se/~nixon/sshprobes.html (cit. on p. 26).

[144] Daiyuu Nobori and Yasushi Shinjo. "VPN Gate: A Volunteer-Organized Public VPN Relay System with Blocking Resistance for Bypassing Government Censorship Firewalls". In: *Networked Systems Design and Implementation*. USENIX, 2014. https://www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-nobori.pdf (cit. on pp. 12, 29, 30, 34).

[145] OpenNet Initiative. *Filtering by Domestic Blog Providers in China*. Jan. 2005. https://opennet.net/bulletins/008/ (cit. on p. 22).

[146] OpenNet Initiative. *Internet Filtering in China in 2004-2005: A Country Study*. https://opennet.net/studies/china (cit. on p. 21).

[147] OpenNet Initiative. *Probing Chinese search engine filtering*. Aug. 2004. https://opennet.net/bulletins/005/ (cit. on p. 22).

[148] Jong Chun Park and Jedidiah R. Crandall. "Empirical Study of a National-Scale Distributed Intrusion Detection System: Backbone-Level Filtering of HTML Responses in China". In: *Distributed Computing Systems*. IEEE, 2010, pp. 315–326. https://www.cs.unm.edu/~crandall/icdcs2010.pdf (cit. on pp. 15, 19).

[149] Vern Paxson. "Bro: A System for Detecting Network Intruders in Real-Time". In: *Computer Networks* 31.23-24 (Dec. 1999), pp. 2435–2463. https://www.icir.org/vern/papers/bro-CN99.pdf (cit. on p. 14).

[150] Paul Pearce, Roya Ensafi, Frank Li, Nick Feamster, and Vern Paxson. "Augur: Internet-Wide Detection of Connectivity Disruptions". In: *Symposium on Security & Privacy*. IEEE, 2017. https://www.ieee-security.org/TC/SP2017/papers/586.pdf (cit. on p. 22).

[151] Paul Pearce, Ben Jones, Frank Li, Roya Ensafi, Nick Feamster, Nick Weaver, and Vern Paxson. "Global Measurement of DNS Manipulation". In: *USENIX Security Symposium*. USENIX, 2017. https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-pearce.pdf (cit. on p. 22).

[152] Mike Perry. *Tor Browser 3.6 is released*. The Tor Blog. Apr. 2014. https://blog.torproject.org/tor-browser-36-released (cit. on p. 10).

[153] Mike Perry. *Tor Browser 4.0 is released*. The Tor Blog. Oct. 2014. https://blog.torproject.org/tor-browser-40-released (cit. on pp. 26, 28, 51, 56).

[154] Mike Perry. *Tor Browser 4.5 is released*. The Tor Blog. Apr. 2015. https://blog.torproject.org/tor-browser-45-released (cit. on pp. 11, 26, 28).

[155] Matthew Prince. *"Thanks for the feedback. . . ."* Hacker News. Mar. 2015. https://news.ycombinator.com/item?id=9234367 (cit. on p. 56).

[156] printempw. 为何 *shadowsocks* 要弃用一次性验证 *(OTA)*. Blessing Studio. Feb. 2017. https://blessing.studio/why-do-shadowsocks-deprecate-ota/. English synopsis at https://groups.google.com/d/msg/traffic-obf/CWO0peBJLGc/Py-clLSTBwAJ (cit. on pp. 25, 26, 28).

[157] *Psiphon*. https://psiphon.ca/ (cit. on p. 50).

[158] Thomas H. Ptacek and Timothy N. Newsham. *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection*. Tech. rep. Secure Networks, Inc., Jan. 1998. https://www.icir.org/vern/Ptacek-Newsham-Evasion-98.pdf (cit. on p. 14).

[159] Abbas Razaghpanah, Anke Li, Arturo Filastò, Rishab Nithyanand, Vasilis Ververis, Will Scott, and Phillipa Gill. *Exploring the Design Space of Longitudinal Censorship Measurement Platforms*. Tech. rep. Oct. 2016. https://arxiv.org/pdf/1606.01979v2.pdf (cit. on p. 22).

[160] *Refraction Networking*. https://refraction.network/ (cit. on p. 13).

[161] Eric Rescorla and Nagendra Modadugu. *Datagram Transport Layer Security Version 1.2*. IETF, Jan. 2012. https://tools.ietf.org/html/rfc6347 (cit. on p. 66).

[162] Hal Roberts, Ethan Zuckerman, and John Palfrey. *2011 Circumvention Tool Evaluation*. Tech. rep. Berkman Center for Internet and Society, Aug. 2011. https://cyber.law.harvard.edu/publications/2011/2011_Circumvention_Tool_Evaluation (cit. on p. 23).

[163] David Robinson, Harlan Yu, and Anne An. *Collateral Freedom: A Snapshot of Chinese Internet Users Circumventing Censorship*. Apr. 2013. https://www.opentech.fund/article/collateral-freedom-snapshot-chinese-users-circumventing-censorship (cit. on p. 53).

[164] Jonathan Rosenberg. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols*. IETF, Apr. 2010. https://tools.ietf.org/html/rfc5245 (cit. on pp. 62, 66).

[165] Jonathan Rosenberg, Rohan Mahy, Philip Matthews, and Dan Wing. *Session Traversal Utilities for NAT (STUN)*. IETF, Oct. 2008. https://tools.ietf.org/html/rfc5389 (cit. on p. 66).

[166] Jonathan Rosenberg and Henning Schulzrinne. *An Offer/Answer Model with the Session Description Protocol (SDP)*. IETF, June 2002. https://tools.ietf.org/html/rfc3264 (cit. on p. 64).

[167] SafeWeb. *TriangleBoy Whitepaper*. http://www.webrant.com/safeweb_site/html/www/tboy_whitepaper.html (cit. on p. 13).

[168] Max Schuchard, John Geddes, Christopher Thompson, and Nicholas Hopper. "Routing Around Decoys". In: *Computer and Communications Security*. ACM, 2012. https://www-users.cs.umn.edu/~hopper/decoy-ccs12.pdf (cit. on p. 13).

[169] Andreas Sfakianakis, Elias Athanasopoulos, and Sotiris Ioannidis. "CensMon: A Web Censorship Monitor". In: *Free and Open Communications on the Internet*. USENIX, 2011. https://www.usenix.org/legacy/events/foci11/tech/final_files/Sfakianakis.pdf (cit. on p. 22).

[170] *Shadowsocks*. https://shadowsocks.org/en/ (cit. on pp. 10, 13).

[171] Charlie Smith. *We are under attack*. GreatFire. Mar. 2015. https://en.greatfire.org/blog/2015/mar/we-are-under-attack (cit. on p. 56).

[172] Rob Smits, Divam Jain, Sarah Pidcock, Ian Goldberg, and Urs Hengartner. "BridgeSPA: Improving Tor Bridges with Single Packet Authorization". In: *Workshop on Privacy in the Electronic Society*. ACM, 2011. https://www.cypherpunks.ca/~iang/pubs/bridgespa-wpes.pdf (cit. on p. 32).

[173] *Snowflake*. Tor Bug Tracker & Wiki. https://trac.torproject.org/projects/tor/wiki/doc/Snowflake (cit. on p. 62).

[174] Qingfeng Tan, Jinqiao Shi, Binxing Fang, Li Guo, Wentao Zhang, Xuebin Wang, and Bingjie Wei. "Towards Measuring Unobservability in Anonymous Communcation Systems". In: *Journal of Computer Research and Development* 52.10 (Oct. 2015). http://crad.ict.ac.cn/EN/10.7544/issn1000-1239.2015.20150562 (cit. on pp. 50, 57).

[175] Tokachu. "The Not-So-Great Firewall of China". In: *2600* 23.4 (2006) (cit. on p. 19).

[176] Tor Metrics. *Bridge users by transport*. Nov. 2017. https://metrics.torproject.org/userstats-bridge-transport.html?start=2017-06-01&end=2017-11-30&transport=obfs3&transport=obfs4&transport=meek&transport=%3COR%3E (cit. on p. 51).

[177] Tor Metrics. *Bridge users by transport from Brazil*. Nov. 2017. https://metrics.torproject.org/userstats-bridge-combined.html?start=2016-06-01&end=2017-11-30&country=br (cit. on pp. 59, 61).

[178] Tor Metrics. *Bridge users from Iran*. Nov. 2017. https://metrics.torproject.org/userstats-bridge-country.html?start=2014-01-01&end=2017-11-30&country=ir (cit. on p. 45).

[179] Tor Metrics. *Bridge users using Flash proxy/websocket*. Dec. 2016. https://metrics.torproject.org/userstats-bridge-transport.html?start=2013-01-01&end=2016-12-31&transport=websocket (cit. on p. 62).

[180] Tor Metrics. *Tor Browser downloads and updates*. Nov. 2017. https://metrics.torproject.org/webstats-tb.html?start=2017-09-01&end=2017-11-30. Source data that gives relative number of stable and alpha downloads is available from https://metrics.torproject.org/stats.html#webstats (cit. on p. 37).

[181] The Tor Project. *BridgeDB*. https://bridges.torproject.org/ (cit. on pp. 12, 37).

[182] Michael Carl Tschantz, Sadia Afroz, Anonymous, and Vern Paxson. "SoK: Towards Grounding Censorship Circumvention in Empiricism". In: *Symposium on Security & Privacy*. IEEE, 2016. https://internet-freedom-science.org/circumvention-survey/sp2016/ (cit. on pp. 6, 9, 18, 23).

[183] Vladislav Tsyrklevich. *Internet-wide scanning for bridges*. tor-dev mailing list. Dec. 2014. https://lists.torproject.org/pipermail/tor-dev/2014-December/007957.html (cit. on p. 25).

[184] *uProxy*. https://www.uproxy.org/ (cit. on pp. 12, 62).

[185] John-Paul Verkamp and Minaxi Gupta. "Inferring Mechanics of Web Censorship Around the World". In: *Free and Open Communications on the Internet*. USENIX, 2012. https://www.usenix.org/system/files/conference/foci12/foci12-final1.pdf (cit. on p. 22).

[186] Liang Wang, Kevin P. Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. "Seeing through Network-Protocol Obfuscation". In: *Computer and Communications Security*. ACM, 2015. http://pages.cs.wisc.edu/~liangw/pub/ccsfp653-wangA.pdf (cit. on pp. 9, 10, 50, 57).

[187] Qiyan Wang, Xun Gong, Giang T. K. Nguyen, Amir Houmansadr, and Nikita Borisov. "CensorSpoofer: Asymmetric Communication using IP Spoofing for Censorship-Resistant Web Browsing". In: *Computer and Communications Security*. ACM, 2012. https://hatswitch.org/~nikita/papers/censorspoofer.pdf (cit. on p. 13).

[188] Qiyan Wang, Zi Lin, Nikita Borisov, and Nicholas J. Hopper. "rBridge: User Reputation based Tor Bridge Distribution with Privacy Preservation". In: *Network and Distributed System Security*. The Internet Society, 2013. https://www-users.cs.umn.edu/~hopper/rbridge_ndss13.pdf (cit. on p. 12).

[189] Zhongjie Wang, Yue Cao, Zhiyun Qian, Chengyu Song, and Srikanth V. Krishnamurthy. "Your State is Not Mine: A Closer Look at Evading Stateful Internet Censorship". In: *Internet Measurement Conference*. ACM, 2017. http://www.cs.ucr.edu/~krish/imc17.pdf (cit. on pp. 15, 26, 28, 43).

[190] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. "StegoTorus: A Camouflage Proxy for the Tor Anonymity System". In: *Computer and Communications Security*. ACM, 2012. https://www.frankwang.org/files/papers/ccs2012.pdf (cit. on p. 10).

[191] Darrell M. West. *Internet shutdowns cost countries $2.4 billion last year*. Oct. 2016. https://www.brookings.edu/wp-content/uploads/2016/10/intenet-shutdowns-v-3.pdf (cit. on p. 9).

[192] Tim Wilde. *CN Prober IPs*. Dec. 2011. https://gist.github.com/twilde/4320b75d398f2e1f074d (cit. on p. 27).

[193] Tim Wilde. *Great Firewall Tor Probing Circa 09 DEC 2011*. Jan. 2012. https://gist.github.com/twilde/da3c7a9af01d74cd7de7 (cit. on pp. 26, 27).

[194] Tim Wilde. *Knock Knock Knockin' on Bridges' Doors*. The Tor Blog. Jan. 2012. https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors (cit. on pp. 26, 27).

[195] Brandon Wiley. *Dust: A Blocking-Resistant Internet Transport Protocol*. Tech. rep. University of Texas at Austin, 2011. http://blanu.net/Dust.pdf (cit. on p. 10).

[196] Philipp Winter. *brdgrd*. 2012. https://github.com/NullHypothesis/brdgrd (cit. on pp. 15, 27).

[197] Philipp Winter. *How the Great Firewall of China is Blocking Tor*. https://www.cs.kau.se/philwint/static/gfc/ (cit. on p. 27).

[198] Philipp Winter. "Measuring and circumventing Internet censorship". PhD thesis. Karlstad University, 2014. https://nymity.ch/papers/pdf/winter2014b.pdf (cit. on p. 6).

[199] Philipp Winter and Stefan Lindskog. "How the Great Firewall of China is Blocking Tor". In: *Free and Open Communications on the Internet*. USENIX, 2012. https://www.usenix.org/system/files/conference/foci12/foci12-final2.pdf (cit. on pp. 8, 15, 20, 26, 27, 31, 34, 43).

[200] Philipp Winter, Tobias Pulls, and Juergen Fuss. "ScrambleSuit: A Polymorphic Network Protocol to Circumvent Censorship". In: *Workshop on Privacy in the Electronic Society*. ACM, 2013. https://censorbib.nymity.ch/pdf/Winter2013b.pdf (cit. on pp. 11, 13, 25).

[201] Sebastian Wolfgarten. *Investigating large-scale Internet content filtering*. Tech. rep. Dublin City University, 2006. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.133.5778&rep=rep1&type=pdf (cit. on p. 19).

[202] Joss Wright, Tulio de Souza, and Ian Brown. "Fine-Grained Censorship Mapping: Information Sources, Legality and Ethics". In: *Free and Open Communications on the Internet*. USENIX, 2011. https://www.usenix.org/legacy/events/foci11/tech/final_files/Wright.pdf (cit. on pp. 2, 18, 34).

[203] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. "Telex: Anticensorship in the Network Infrastructure". In: *USENIX Security Symposium*. USENIX, 2011. https://www.usenix.org/event/sec11/tech/full_papers/Wustrow.pdf (cit. on p. 49).

[204] Xueyang Xu, Z. Morley Mao, and J. Alex Halderman. "Internet Censorship in China: Where Does the Filtering Occur?" In: *Passive and Active Measurement Conference*. Springer, 2011, pp. 133–142. https://web.eecs.umich.edu/~zmao/Papers/china-censorship-pam11.pdf (cit. on p. 20).

[205] *XX-Net*. https://github.com/XX-net/XX-Net (cit. on p. 60).

[206] Yawning Angel and Philipp Winter. *obfs4 (The obfourscator)*. May 2014. https://gitweb.torproject.org/pluggable-transports/obfs4.git/tree/doc/obfs4-spec.txt (cit. on pp. 11, 13, 25, 35).

[207] Tao Zhu, David Phipps, Adam Pridgen, Jedidiah R. Crandall, and Dan S. Wallach. "The Velocity of Censorship: High-Fidelity Detection of Microblog Post Deletions". In: *USENIX Security Symposium*. USENIX, 2013. https://www.cs.unm.edu/~crandall/usenix13.pdf (cit. on p. 34).

[208] Jonathan Zittrain and Benjamin G. Edelman. "Internet filtering in China". In: *IEEE Internet Computing* 7.2 (Mar. 2003), pp. 70–77. https://dash.harvard.edu/bitstream/handle/1/9696319/Zittrain_InternetFilteringinChina.pdf (cit. on p. 18).

Index