Draft of November 20, 2017

Chapter 1

Introduction

This is a thesis about Internet censorship. In it, I will expand on two threads of research that have occupied my attention for the past several years: better understanding how censors work, and fielding systems that circumvent their restrictions. These two threads fuel each other: better understanding censors enables us to build better circumvention systems that take into account their strengths and weaknesses; and the deployment of a circumvention system affords an opportunity to observe how censors themselves react to changing circumstances. If I am successful, the output of my research is useful models that describe not only how censors behave today but how they may evolve in the future, and tools for circumvention that are not only sound in theory but also effective in practice.

1.1 Scope

Censorship is an enormous topic. Even the addition of the "Internet" qualifier hardly reduces its scope, because almost everything that might be censored touches the Internet in some way. To deal with the subject in depth, it is necessary to limit the scope. My research is focused on an important special case of censorship, which I call the "border firewall" case. It is illustrated in Figure 1.1.

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

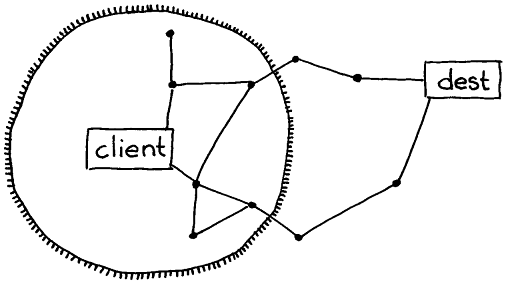

Figure 1.1: In the border firewall scenario, a client within a censor-controlled network wants to reach a destination that lies outside the censor's control.

any link. The censor, in particular, tries to prevent some subset of communication with the wider Internet, for instance by blocking certain keywords, network addresses, or protocols. The client's computer, however, is trustworthy and not controlled by the censor. The client's goal is to communicate with some *destination* that lies outside the censor's network, despite the censor's blocks: we call this activity *circumvention*. Circumvention requires somehow safely traversing a hostile network, eluding detection and blocking by the censor, in order to reach a destination. The censor does not control network links outside its own border; it may of course 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 the situation in practice is never so clear-cut. For example, the censor may be weaker than assumed: it may observe only the links at the border, not those wholly inside; it may not be able to fully inspect every packet that flows on its network; or there may be deficiencies or dysfunctions in its detection capabilities. Or the censor may be stronger: perhaps it, while not fully controlling outside networks, may influence their operators to discourage them 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, or may not. There is no limit to the possible complications. Adjusting the basic model to reflect real-world actors' motivations and capabilities is the heart of threat modeling, one of the main topics of this thesis. Depending on the situation, we will add to the list of assumptions. In particular, what makes circumvention possible at all is the censor's motivation to block only *some* of the incoming and outgoing traffic, and allow the rest to pass—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 a national government, and the borders of its controlled network correspond to the borders of a country. A government typically has the power to enforce laws and control network infrastructure to act within its own borders, but not outside. However the boundaries of censorship do not always correspond exactly to the border of a country. Almost since the study of Internet censorship began, it has been recognized that content restrictions may vary across geographic locations, even within the same country (Wright et al. [153] identified some possible causes). In some places a better model is not a single unified censorship regime, but rather many individual Internet service providers, each controlling its own network and acting as a mini-censor, 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 some 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
- dissecting network layers (“deep packet inspection”)
- statistical and probabilistic traffic classification
- connection speed throttling
- active measures by censors to discover the use of circumvention

Other forms of censorship that are *not* in scope include:

- domain takedowns (that affect all clients globally)
- server-side blocking (servers refusing to serve certain clients)
- anything that takes place entirely within the censor’s network and does not cross the border
- forum moderation and deletion of social media posts
- deletion-resistant publishing in the vein of the Eternity Service [6] (what Köpsell and Hillig call “censorship resistant publishing systems” [86 §1]), except insofar as access to such services may be blocked

Many 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, and softer forms of disruption such as bandwidth throttling. Detection need not be purely passive. The censor is permitted 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 entities inside and outside the censor’s network, and indeed almost all circumvention will require the cooperation of a willing proxy on the outside.

Some have objected to the use of the generic term “Internet censorship” to refer to the narrow case of the border firewall. I am sensitive to this objection and acknowledge that far more topics could fit under the umbrella of Internet censorship. Nevertheless, for the purpose of this thesis, I will continue to use “Internet censorship” without further qualification to refer to the border firewall case.

1.2 Context and overview

This thesis contains knowledge I have collected and research projects I have taken part in over the last five years. The next chapter, “Principles of circumvention,” is the thesis of the thesis, wherein I lay out opinionated general principles of the field. The remaining chapters are split between the topics of modeling and circumvention: Chapters 3–5 on censor modeling and Chapters 6 and 7 on circumvention systems.

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. 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. whose anonymity network has been the vehicle for deployment of my circumvention systems, as well as a common object of research. 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 a language 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

- Pluggable transports

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. A censor must be able both to detect and to block. (Detection without blocking would be called surveillance, not censorship.) The flip side of this statement is that a circumventor succeeds either by eluding detection, or, once detected, somehow resist 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 it 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 for avoiding detection are often called network protocol "obfuscation," and the term is apt. 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 given censor and its specific resources and motivations. Understanding the relative importance the censor assigns to classification errors—knowing what it prefers to allow and to block—is helpful. Through good modeling, we can make the tradeoffs less favorable for the censor and more favorable for the circumventor.

The censor may base its detection decision on whatever criteria it find 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. Of course, there is no clear separation between what is content and what is an address. The layered nature of network protocols means that an address at one layer is content at another. 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 intermediary 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. Throughout this thesis, I will use the word "proxy" with an abstract meaning of "one that acts of behalf of another." A proxy need not be what is typically understood by the term "proxy server," a single host accepting and forwarding connections. A VPN (virtual private network) is also a kind of proxy, as is the Tor network, as may be a specially configured network router. In Chapter 6 we will see a network of cloud servers acting as a proxy. In Chapter 7 the proxy will be a pool of temporary instances of some JavaScript code.

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 access to a proxy, which transitively gives access to the destination. The censor attempts deals 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 way. Köpsell and Hillig divide detection into "content" and "circumstances" [86 §4]; their circumstances include addresses and also what I would consider more content-like: timing, data transfer characteristics, and protocols. Philipp Winter divides circumvention into three problems: bootstrapping, endpoint blocking, and traffic obfuscation [148 §1.1]. 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., in their 2016 survey and systematization of circumvention systems, break detection into four aspects: destinations, content, flow properties, and protocol semantics [84 §2.4]. I think of their "content," "flow properties," and "protocol semantics" as all fitting under the heading of content. Tschantz et al. identify "setup" and "usage" [135 §V], and Khattak, Elahi, et al. identify "communication establishment" and "conversation" [84 §3.1], as targets of obfuscation; these mostly correspond to address and content. What I call "detection" and "blocking," Khattak, Elahi, et al. call "fingerprinting" and "direct censorship" [84 §2.3], and Tschantz et al. call "detection" and "action" [135 §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 stress 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 browser, and work out details of 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 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 weak circumvention system, watching it get blocked as it becomes popular, and starting over again with another similarly weak system. In my opinion, the cat-and-mouse game is not inevitable. It is possible to develop systems that resist blocking—not absolutely, but quantifiably in terms of costs to the blocker—even after it has become popular. We should think of the honeymoon period while a system is too small to be worth noticing, not as the beginning and end of a system's useful life, but as a time to work out growing pains.

2.1 Collateral damage

What's to prevent the censor from shutting down all connectivity within its network, trivially preventing the client from reaching the destination? The answer is that the censor derives some kind of benefit from allowing network connectivity, other than that which it tries to censor. Or to put it another way: the censor *incurs a cost* whenever it commits a false positive (also called overblocking: inadvertently blocking 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 fully reach this ideal 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 for why the Great Firewall of China has never blocked GitHub for long, despite GitHub's hosting and distribution of circumvention software: GitHub is so deeply integrated into software development, that programmers are not able to work when it is blocked.

when and how long?

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.

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 reducing false negatives.) For example, it has been repeatedly documented—by Clayton et al. [19], Winter and Lindskog [149], and Fifield and Tsai [66], for example—that the Great Firewall prefers to block individual ports (or a small range of ports), rather than blocking an entire IP address, probably in a bid to reduce collateral damage. In Chapter 6 we will see a system whose blocking resistance is based on widely used web services—the argument is that to block the circumvention system, the censor would have to block the entire web service. However this argument requires that the circumvention system's use of the web service be indistinguishable from other uses—otherwise the censor may selectively block only the connections used for circumvention. 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 is not capable of blocking.) To deploying a circumvention system is to make a bet: that the censor cannot field a classifier that adequately distinguishes 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 a collateral damage argument: that the censor feels that to block that other protocol would result in too much damage to be worth it. For example, a circumvention 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 the circumvention protocol has no "tell" that enables the censor easily 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 of no use to have a perfect steganographic 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 is able to take 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 have been many other attempts at defining resistance to blocking. Narain et al. [106] called the essential element "deniability," meaning that a user could plausibly claim to have been doing something other than circumventing when confronted with a log of their network activity. Khattak, Elahi, et al. [84 §4] also consider "deniability" separately from "unblockability." Houmansadr et al. [75, 76, 77] used the term "unobservability," which I feel fails to convey that the censor's essential function is distinguishing, not observation. Brubaker et al. [12] used the term "entanglement," which is closer to the mark and inspired my own thinking. What they call entanglement I think of as indistinguishability, and keep in mind that that which you are trying to be indistinguishable with has to be something 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 doing Y and others may not; there is no "unblockable" in absolute terms. Now, actually quantifying the value of Y is a task in itself, by no means a trivial one. The state of research in this field is still far from being able to assign actual numbers (e.g. in terms of dollars) to costs as perceived 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 do complete shutdowns. In fact the practice is increasing; reported of shutdowns in 2016. This does not necessarily contradict the theory of collateral damage. Shutdowns are indeed costly—estimated that shutdowns cost . It is just that, in some cases, the calculus works out that the harm caused by a shutdown does not outweigh (in the censor's mind) the benefits of blocking access. 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 or "curfews" are limited in duration shows that even censors that can afford to do a total shutdown cannot afford to do it forever.

someone

some number

someone

some amount

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. One might even say that the very decision to censor is exactly such an irrational decision, at the greater societal level.

2.2 Content obfuscation strategies

- Sony thing on passive/active detection [128 §5.1]
- relation to website fingerprinting—circumvention is potentially harder because you can't just use e.g. constant bitrate

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

Tschantz et al. [135] call these two strategies "steganography" and "polymorphism" respectively. Another way to say it is "look like something" and "look like nothing." They are not strict classifications—any real system will incorporate a bit of both—and 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, falls to 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 effectiveness can only be judged against a censor's specific computations of 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. [75] exhibited weaknesses in "parrot" circumvention systems that mimic a cover protocol but do not perfectly imitate it. 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. [139] found that some of Houmansadr et al.'s proposed attacks were impractical, due to high false-positive rates, but proposed other attacks designed for efficiency and low false positives, against both steganographic and polymorphic protocols. Geddes et al. [68] showed that even perfect imitation (achieved via tunneling) may leave vulnerabilities due to mismatches between the cover protocol and the covert protocol—for instance randomly dropping packets may disrupt circumvention more than other uses 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 literature have not been used by censors in practice.

Some systematizations (for example those of Brubaker et al. [12 §6]; Wang et al. [139 §2]; and Khattak, Elahi, et al. [84 §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 useful, except when speaking of concrete implementation choices; to me, there are various degrees of fidelity in imitation, and tunneling only tends to offer higher fidelity than mimicry.

I will list some representative circumvention systems that exemplify the steganographic strategy. Infranet [45], way back in 2002, built a covert channel out of HTTP, encoding upstream data in special requests and downstream data using standard steganography in image files. (An aside on the evolution of threat models: the authors of Infranet rejected the possibility of using TLS (then called SSL), because it was not then common enough that its wholesale blocking would cause much damage. Today the situation around TLS is much different, and it is much relied on by circumventors.) StegoTorus [142] (2012) uses custom encoders to make traffic resemble common HTTP file types, such as PDF, JavaScript, and Flash. SkypeMorph [103] (2012) mimics a Skype video call. FreeWave [77] (2013) modulates a data stream into an acoustic signal and transmits it over VoIP. FTE [41] (for "format-transforming encryption"; 2013) and its followup Marionette [42] (2015) force traffic to conform to a user-specified syntax: if you can describe it, you can imitate it. Despite the research attention they have received, steganographic systems have not been as used in practice: of these listed systems, FTE is the only one that has seen substantial deployment.

There are many examples of the randomized, polymorphic strategy. An important subclass of these are 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 [87], which aimed to hide the plaintext packet metadata in the SSH protocol. obfuscated-openssh worked, in essence, by first sending a cryptographic key, then sending ciphertext encrypted with that key. The encryption of the obfuscation layer was an additional, independent layer on top of SSH's usual encryption. A censor could, in principle, purely passively detect and deobfuscate the protocol just by recovering the key and using it to decrypt the rest—a situation partially mitigated by the use of an expensive key derivation function based on iterated hashing. obfuscated-openssh could optionally incorporate a pre-shared password into the key derivation function, which would prevent easy identification. Dust [146], a design by Brandon Wiley, 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, relying on an out-of-band key exchange before each session. Shadowsocks [127] is a lightweight encryption layer atop a simple proxy protocol.

There is a line of successive look-like-nothing protocols—known by the names obfs2, obfs3, ScrambleSuit, and obfs4—whose history is interesting, because it illustrates mutual advances by censors and circumventors over several years. obfs2 [82], which debuted in 2012 in response to blocking in Iran [28], uses very simple obfuscation inspired by obfuscated-openssh: it is essentially equivalent to sending an encryption key, followed by the rest of the stream encrypted by 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 the proxy to see what protocol it uses. However, it was sufficient against the keyword- or pattern-based censors of its era. obfs3 [83]—first available in 2013 but not really released to users until 2014 [115]—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 February 2015, or possibly as early as July 2013 [43 §5.4].) ScrambleSuit [150], first available to users in 2014 [17], arose in response to the active-probing of obfs3. Its improvements 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, or else the server will not respond, preventing active probing. (Active probing resistance really has more to do with blocking by address than with blocking by content, but it is only because the randomized transports sufficiently frustrated content-based detection that active probing became relevant.) obfs4 [156], first available in 2014, 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 lengths and timing to fit random distributions. Crucially, the chosen distributions are consistent within each server, not generated afresh for each connection. That means that even if a censor is able to build a profile for a particular server, it is not necessarily useful for detecting other server instances.

2.3 Address blocking resistance strategies

- VPN Gate “collaborative spy detection” [110 §4.3], other ways of fingerprinting censor
- DEFIANCE [90]

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, from a circumvention point of view—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 what it sends, where it can be observed by the censor, if the encoding is sufficiently transparent. The second is the insider attack: legitimate clients must have some way to discover addresses of, e.g., 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: private proxies shared by 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; and 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 [34 §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 [137] 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 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 them all to the censor. Both of these considerations pose challenges.

Tor’s blocking resistance design [34], based on secret proxies called “bridges,” was of this kind. Volunteers run bridges, which report themselves to central database called BridgeDB [134]. 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 easy enumeration of the entire database [92]. 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 permitted 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 an exploit allowing enumeration of one distribution method does not affect the others. But even these defenses may not be enough: despite public appeals for volunteers to run bridges (see for example Dingledine’s initial call in 2007 [29]), there have never been more than a few thousand of them, and Dingledine reported in 2011 that the Great Firewall of China had managed to enumerate both the HTTPS and email distribution pools [30 §1, 31 §1], presumably taking advantage of its greater resources.

Tor relies on BridgeDB to provide address blocking resistance for all its transports that otherwise only have 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 plausibly 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 “bridge distribution problem” or “proxy discovery problem.” I will give just a summary of various proposals.

Short summaries of proxy distribution papers.

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, blocking which would result in high collateral damage. VPN Gate employed this idea [110 §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 [31 §6, 34 §9.3]. Durumeric et al. found about 80% of existing (unobfuscated) Tor bridges [40 §4.4] by scanning all of IPv4 on a handful of common bridge ports. Matic et al. had similar results in 2017 [99 §V.D], using public search engines in lieu of active scanning. The best solution to the scanning problem is to do as ScrambleSuit and obfs4 do, and associate with each proxy a secret, without which a client cannot initiate a connection. The critical part is that the IP address and port must not constitute the whole of the information needed to connect to the proxy. Scanning for bridges is closely related to active probing, the topic of Chapter 4.

Better understanding of Kaleidoscope.

An alternative 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 [60] and Snowflake [131]. (Snowflake is the topic of Chapter 7.) Even proxy distribution strategies that take churn into account have in mind proxies that last on the order of at least 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. Flash proxy and Snowflake proxies have a low set-up and tear-down cost: you can run one just by visiting a web page. These designs do not to need a sophisticated proxy distribution strategy as long as the rate of proxy creation is kept higher than the censor’s rate of discovery.

Enemy at the Gateways [107]

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 such high collateral damage that the censor does not dare block them. Refraction networking [121], also called decoy routing, puts proxy capability into network routers—in the middle of paths, rather than at the end. Clients 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. The censor has to induce routes that avoid the special routers [125], which is costly [78]. Domain fronting [65] 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 [140]. 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 [124] employed lighter-weight intermediate proxies that would simply forward 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, spoof its responses so they appeared to originate from the intermediate 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

- Deniable Liaisons [106]

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 not block it, Sometimes it is useful to consider this possibility when modeling. Khattak, Elahi, et al. [84] 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, 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. [19]. 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 client to become unsynchronized), and that simply ignoring the anomalous packets rendered blocking ineffective. (Why then, did the censor choose to *inject* its own packets, rather than *drop* the client's or server's? 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 effects blocking. Additionally, intensive measurements revealed many failures to block, and blocking rates that changed over time, suggesting that even when the firewall intends a general 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 [108], a file distribution service based on satellite TV 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 Toosheh 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 [120] and Paxson [114 §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 able it to predict exactly how each packet will be interpreted by the endpoints. Cronin et al. [24] posit that the monitor's conflicting goals of 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—eliminating certain traffic features from its consideration. Crandall et al. [21] in 2007 suggested using IP fragmentation to prevent keyword matching (splitting keywords across fragments). In 2008 and 2009, Park and Crandall [113] 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 [149] found that the Great Firewall still did not do TCP segment reassembly in 2012, in the course of studying the firewall's proxy-discovery probes. (Such probes are the subject of Chapter 4.) They released a tool, brdgrd [147], 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. They reported that the tool stopped working in 2013. Anderson [5] 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. [85] 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.

Your State is Not Mine [141]

2.5 Early censorship and circumvention

Internet censorship and circumvention began to rise to importance in the mid-1900s, coinciding with the popularization of the World Wide Web. At that time, online censorship focused mainly on the web. Computer security companies were developing technology for IP address, URL, and web page filtering. 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 [102] reported in 1996 on the secret blocking lists of several programs. Bennett Haselton and Peacefire [73] 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. The term "Great Firewall of China" first appeared in an article in *Wired* magazine [9] in 1997. In some cases adapting the same technology originally developed for personal firewalls. 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 URL on behalf of the client, and before returning the response, rewrite all the links and external references in the page to make the relative to the proxy. CGIProxy [97], SafeWeb [98], Circumventor [72], and the first version of Psiphon [16] were all of this kind.

These systems were effective against their censors of their day—at least with respect to destination blocking. And 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 [34 §7.4] which would send out fresh proxy addresses every few days. A 1996 article by Rich Morin [104] 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 clearly did 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 begin to experiment with censorship, such as in Turkey in 2014, where alternative DNS servers briefly sufficed to circumvent a block of Twitter [23].) Not only censors were changing—the world around them was changing as well. In this field that 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, in 2001, that TLS would not suffice as a cover protocol [45 §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

Here be dragons

A detached view is helpful when taking a longer view. (As long as it is not *too* detached.)

The main tool we have to build relevant threat models is the natural 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 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. What we are really interested in, and what is most difficult to infer, is the *why*, or the motivations and goals that underlie a censorship apparatus. We posit that censors, far from being unitary entities of focused purpose, are rather complex organizations of people and machines, with conflicting purposes and economic rationales, subject to resource limitations. The *why* gets to the heart of why circumvention is even possible: a censoring firewall's duty is not merely to block, but to *discriminate* between what is blocked and what is allowed, in support of some other goal. Circumvention systems confuse this discrimination in order to sneak traffic through the firewall.

Past measurement studies have mostly been short-lived, one-off affairs, focusing deeply on one region of the world for at most a few months. Thus published knowledge about censors' capabilities consists mostly of a series of "spot checks" with blank areas between them. There have been a few designs proposed to do ongoing measurements of censorship, such as ConceptDoppler [21] in 2007 and CensMon [126] in 2011, but these have not lasted long in practice, and for the most part there is an unfortunate lack of longitudinal and cross-country measurements. Just as in circumvention, in censorship measurement a host of difficulties arise when running a scalable system for a long time, that do not arise when doing a one-time operation. The situation is thankfully becoming better, with the increasing data collection capabilities of measurement systems like OONI [67].

From the survey of measurement studies we may draw some general conclusions. Censors change over time, sometimes for unknown reasons, and not always in the direction of greater restrictions. Censorship conditions differ greatly across countries, not only in subject but in mechanism and motivation. The "Great Firewall" of China has long been the world's most sophisticated censor, but it is in many ways an outlier, and not representative of censors elsewhere. Most censors are capable of manipulating DNS responses, IP address blocking, and keyword filtering at some level.

A reasonable set of capabilities, therefore, that a contemporary censor may be assumed to have is:

- blocking of specific IP addresses and ports,
- control of default DNS servers,
- injection of false DNS responses,
- injection of TCP RSTs,
- throttling of connection,
- keyword filtering
- protocol identification, and
- temporary total shutdown of Internet connections

Not all censors will be able 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 censoring act even makes sense will depend on a local cost–benefit analysis. Some censors may be able to tolerate a brief total shutdown, while for others the importance of the Internet is too great for such a crude measure.

Past measurement studies have done a good job at determining the technical aspects of censorship, for example where in the network censorship routers are located. There is not so much known about the inner workings of censors. The anonymous paper on China's DNS censorship [7] probably comes closest to the kind of insight I am talking about, with its clever use of side channels to infer operational characteristics of censor boxes. For example, their research found that each DNS injection node runs a few hundred independent processes. This is indirect information, to be sure, but it hints at the level of resources the censor is able to bring to bear. I am interested in even deeper information, for example how censors make the decision on what to block, and what bureaucratic and other considerations might cause them to work less than optimally.

informing our threat models

censors' capabilities—presumed and actual e.g. ip blocking (reaction time?) active probing

Internet curfews (Gabon), limited time of shutdowns shows sensitivity to collateral damage.

commercial firewalls (Citizen Lab) and bespoke systems

Ongoing, longitudinal measurement of censorship remains a challenge. Studies tend to be limited to one geographical region and one period of time. Dedicated measurement platforms such as OONI [67] and ICLab [79] are starting to make a dent in this problem, by providing regular measurements from many locations worldwide. Even with these, there are challenges around getting probes into challenging locations and keeping them running.

Apart from a few reports of, for example, per annum spending on filtering hardware, not much is known about how much censorship costs to implement. In general, contemporary threat models tend to ignore resource limitations on the part of the censor.

Tying questions of ethics to questions about censor behavior, motivation: [153] (also mentions "organisational requirements, administrative burden") [80] [20] Censors may come to conclusions different than what we expect (have a clue or not).

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.

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 [122] and Freedom House [14] in 2011. Rather than testing tools against censors, we evaluated how closely calibrated 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 bridge distribution and rendezvous. Indeed, in our survey of over 50 circumvention tools, we found that academic designs tended to be concerned with detection in the steady state after a connection is established, while actually deployed systems cared more about how the connection is established initially. 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.

This work on evaluation appeared in the 2016 research paper "Towards Grounding Censorship Circumvention in Empiricism" [135], which I coauthored with Michael Carl Tschantz, Sadia Afroz, and Vern Paxson.

Do they check the right things?

what's used and what's not used

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 the destination will be blocked in the future. The input to active probing, a set of suspected addresses, comes from passive observation of the content of client connections. The censor sees a client connect to a destination. Whenever the censor's content classifier is unsure whether an ongoing connection is accessing a proxy, it may pass the address of the destination to the active prober. The active prober's connection then checks—with a low chance of false positives—whether the destination 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 the precision, by only blocking those servers determined through active inspection to be proxies. With active probing, the censor can get away with a mediocre content-based classifier, one that returns a rough superset of actual proxy connections, because active probes will weed out any false positives it might have had. The content-based classifier only has to reduce the total connections to a small enough number that the active probing subsystem can handle them. Another benefit, from the censor's point of view, is that active probing can be run as a batch job, separate from the the firewall's responsibilities that require a low response time.

Active probing, as I use the term in this chapter, is distinguished 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 on 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 [34 §9.3] raised scanning resistance as an issue in Tor's initial bridge design document. McLachlan and Hopper [100 §3.2] observed that the tendency of bridges to run on a handful of popular ports would make them discoverable in an Internet-wide scan, which they estimated would take weeks. Dingledine [31 §6] mentioned indiscriminate scanning as one of ten ways to discover Tor bridges—while also bringing up the possibility of active probing in the sense of the present chapter, then just beginning to placeholder for figure

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

be used by the Great Firewall. Durumeric et al. [40 §4.4] demonstrated the effectiveness of Internet-wide scanning, discovering about 80% of public bridges in a matter of hours, targeting only two ports, 9001 and 443. Tsyrklevich [136] and Matic et al. [99 §V.D] later showed how to existing public repositories of Internet scan data could reveal many bridges, without even the necessity of manually running a new scan.

The Great Firewall of China is the only censor known to employ active probing. Its sophistication has increased over time, with the addition of new protocols and a reduction in the delay before new servers get probed. The Great Firewall has the documented ability to active-probe plain Tor and some of its pluggable transports, certain VPN protocols, as well as certain HTTPS-based proxies. The 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 lies somewhere in the middle of the dichotomy, put forward in Chapter 2, of blocking by content and blocking by address. The censor's active probing subsystem takes addresses as input and produces addresses as output (to be added to a blacklist). But it is content-based classification that produces the list of input addresses. Active probing only became an issue because content obfuscation had gotten better: if a censor could easily identify circumvention protocols by passive inspection, it would not go to the extra trouble of active probing.

Contemporary circumvention systems must be designed to resist active probing attacks. The look-like-nothing systems ScrambleSuit [150], obfs4 [156], and Shadowsocks [94, 119] do it by having the proxy authenticate client connections, using a per-proxy password or private key. Domain fronting (Chapter 6) and Snowflake (Chapter 7) deal with active probing differently.

2010 August	Nixon notices strange, random-looking connections from China in his SSH logs [109].
2011 May–June	Nixon's random-looking probes are temporarily replaced by TLS probes before changing back again [109].
2011 October	hrimfaxi reports that Tor bridges are quickly detected by the GFW [27].
2011 November	Nixon publishes observations and hypotheses about the strange SSH connections [109].
2011 December	Tim Wilde investigates Tor probing [33, 144, 145]. He finds two kinds of probe: "garbage" random probes and Tor-specific ones.
2012 February	The obfs2 transport becomes available [28]. The Great Firewall is initially unable to active-probe it.
2012 March	Winter and Lindskog investigate Tor probing in detail [149].
2013 January	The Great Firewall begins to active-probe obfs2 [32, 43 §4.3]. The obfs3 transport becomes available [48].
2013 June–July	Majkowski observes TLS and garbage probes and identifies fingerprintable features of the probers [95].
2013 August	The Great Firewall begins to active-probe obfs3 [43 Figure 8].
2014 August	The ScrambleSuit transport (resistant to active probing) becomes available [116].
2015 April	The obfs4 transport (resistant to active probing) becomes available [117].
2015 August	BreakWa11 discovers an active-probing weakness in ShadowSocks [10, 119 §2].
2015 October	Ensafi et al. [43] publish results of multi-modal experiments on active probing.
2017 February	Shadowsocks changes its protocol against active probing [74].

Table 4.1: Timeline of active probing research.

4.1 History of active probing research

Active probing research has primarily had to do with Tor and its pluggable transports. There is also some work on Shadowsocks. Table 4.1 summarizes the research of this section.

Nixon [109] in late 2011 published an analysis of suspicious connections from IP addresses in China that his servers had 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 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, sending non-protocol-conforming data to see how the server would respond. For a few weeks in May and June 2011, the probes did not look random, but 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 first being accessed from China [27]. Moving the bridge's address to another port on the same IP address would work temporarily, but then be blocked again before another 10 minutes. Wilde systematically investigated the phenomenon and published an extensive analysis of active probing behavior caused by making a connection from inside China to outside [144, 145]. 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 sent in response to TLS connections to port 443 only, and followed the triggering connection within moments. The Tor probes were sent in response to TLS connections on any port that shared characteristics with Tor’s client handshake [33], and were not sent immediately, but batched to the next quarter hour. The probes came from diverse IP addresses in China: 20 different /8 networks [143]. Bridges using the obfs2 transport were neither probed nor blocked.

Winter and Lindskog revisited the question of Tor probing a few months later in 2012 [149]. They used open proxies and a VPS in China to reach bridges and relays in Russia, Singapore, and Sweden (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. Blocks expired after 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 source IP addresses of probes after a few minutes sometimes found a live host, though usually with a different IP TTL than was used during the probing, which the authors suggest 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 [147], 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.

The obfs2 pluggable transport, first available in February 2012 [28], worked against active probing for about a year. The first report of its active probing arrived in March 2013 [32]. By analyzing the logs of my web server, I found evidence for an even earlier onset: January 2013 [43 §4.3]. At about the same time, the obfs3 pluggable transport became available [48]. It was as vulnerable to active probing as obfs2 was, but the firewall did not gain the ability to active-probe it until August 2013 [43 Figure 8].

Majkowski [95] observed a change in active-probing behavior between June and July 2013. In June, he reproduced the observations of Winter and Lindskog, eliciting pairs of TLS probes, one from 202.108.181.70 and one from another IP address. He also provided TLS fingerprints for the probers, which were distinct from the fingerprints 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 probes in July differed from those seen earlier, but were still identifiable.

The ScrambleSuit transport, designed to be immune to active-probing attacks, first shipped with Tor Browser 4.0 in October 2014 [116]. The successor transport obfs4, similarly immune, shipped in Tor Browser 4.5 in April 2015 [117].

In August 2015, developer BreakWa11 described an active-probing vulnerability in the Shadowsocks protocol [10, 119 §2]. The flaw had to do with a lack of authentication of ciphertext, allowing a prober to introduce errors and watch how the server responds. The Shadowsocks developers deployed a modified protocol, a stopgap measure that proved to have its own vulnerabilities to probing. Shadowsocks deployed another protocol change in

February 2017 fixing the problem [74]. Despite the long window of vulnerability, there is no evidence that the Great Firewall tried to active-probe Shadowsocks servers [112].

Ensafi et al. (including me) [43] 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 back-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 is capable of sending, the probers' source addresses, and potentially fingerprintable peculiarities of the probers' implementation of protocols. Observations from this research project appear in the remaining sections of this chapter.

4.2 Types of probes

Our experiments confirmed the existence of certain probe types that had been documented in previous research, and other types that had not been previously documented. Of the probe types that had been documented before, our observations were mostly consistent, with some differences in the details. Our research found, at varying times, these kinds of probes:

Tor We expected to find probing for Tor, and so we did. The probes we observed in 2015, however, differed from those Wilde described in 2011: ours had a lighter-weight check inside the TLS layer that did not require building a circuit. Also, in contrast to what Winter and Lindskog found in 2012, our Tor probes were sent seconds after a connection, no longer batched to a multiple of 15 minutes.

obfs2 The obfs2 protocol has a weakness that makes it trivial to identify, passively or retroactively, as long as you have at least the first 20 bytes sent by the client. We turned the weakness to our advantage. The ready identifiability of obfs2 allowed us to distinguish it from other random-looking contents and isolate a set of connections that could only belong to legitimate circumventors or active probers.

obfs3 Unlike obfs2, the obfs3 protocol is not easily identified passively, except by general characteristics like its random payloads and certain bounds on message sizes during the initial handshake. In certain of our experiments, we were running an obfs3 server that was able to participate in the handshake and so confirm that what was being sent was really obfs3. In others, such as the passive log analysis, we called "obfs3" those probes that looked random and were not obfs2.

SoftEther We were initially only looking for Tor-related active probing, but in the process we inadvertently found other kinds of probes. One of these was an HTTPS request, "`POST /vpnsvc/connect.cgi HTTP/1.1`", which resembles the client handshake of the SoftEther VPN software that underlies the VPN Gate circumvention system [110].

AppSpot This type of probe is an HTTPS request,

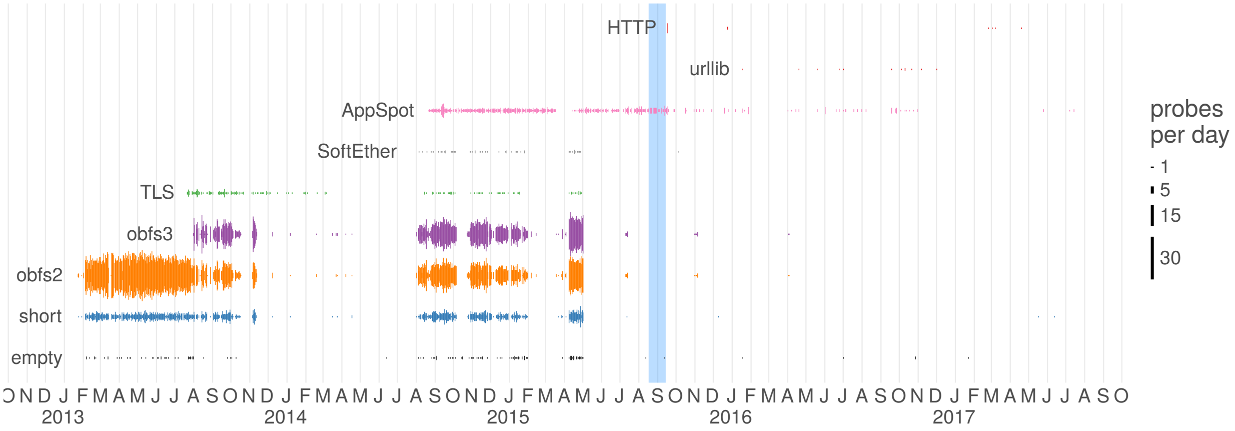

Figure 4.2: Active probes received at my web server over five years. This is an updated version of Figure 8 from the paper "Examining How the Great Firewall Discovers Hidden Circumvention Servers" [43]; the vertical blue stripe divides old and new data. The "short" probes are those that looked random but did not provide enough data (20 bytes) for the obfs2 test; it is likely that they, along with the "empty" probes, are really obfs2, obfs3, or Tor probes that were truncated at the first '\0' or '\n' byte.

urllib

Active probing activity—at least against this server—has subsided since 2016.

```
GET / HTTP/1.1
Host: webncsproxyXX.appspot.com
```

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. (See Chapter 6 for more on domain fronting.) At one time, there were simple proxies running at webncsproxyXX.appspot.com.

urllib describe; this one is new since the 2015 paper

This is not an exhaustive list of the Great Firewall's active probing capability; these are just the probes we were able to document comprehensively. The purpose of the random "garbage" probes that Nixon and Wilde had described is still not known; they were not obfs2 and were too early to be obfs3, so they must have been something else.

Most of our experiments were designed around exploring known Tor-related probe types: plain Tor (without pluggable transports), 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/mail server, which was also a Tor bridge. Application-layer logs lack much of the fidelity you would 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 a '\0' or '\n' byte. But they make up for all that with time coverage. Section 4.2 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 started by searching the logs for likely

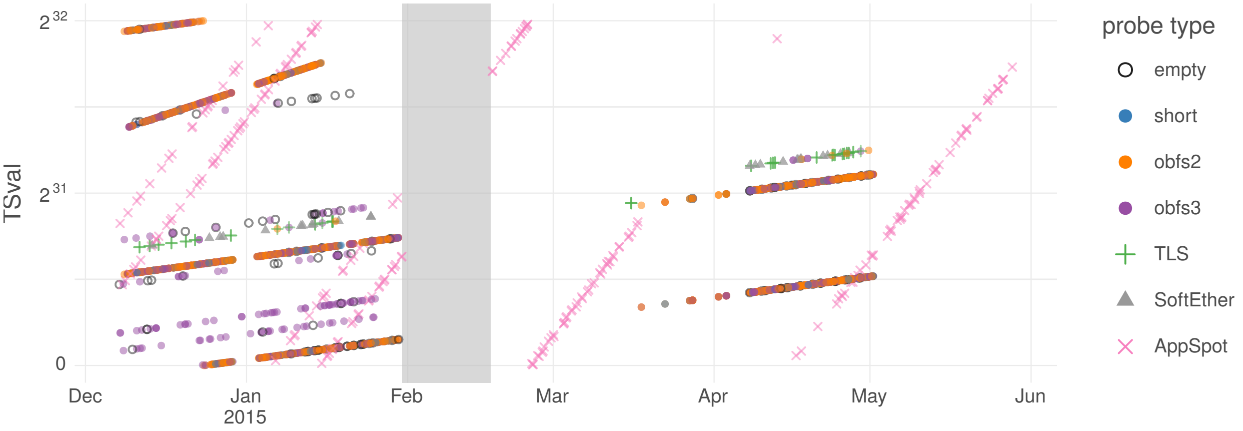

Figure 4.3: TCP timestamp values from active probes. Depicted are 4,239 probes from 3,797 distinct source IP addresses, sharing however only a few TCP timestamp sequences. The shaded area ■ marks a gap in packet capture.

probes: those that passed the obfs2 test or otherwise looked like random garbage. Then we looked at what else appeared in the logs for the IP addresses that had sent the certain probes. 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.

4.3 Probing infrastructure

The most salient feature of active probes, when considered all together, is the large number of source addresses from which they are sent. The 13,089 probes received by the HTTP and HTTPS ports of my server came from 11,907 distinct IP addresses, 96% of them appearing only once. There is one extreme outlier, the address 202.108.181.70, which by itself accounted for 2% of the probes. Among the address ranges are ones belonging to residential ISPs.

Despite these many source addresses, the sending of probes seems to be controlled by only a few underlying processes. The evidence for this lies in shared metadata patterns: TCP initial sequence numbers and TCP timestamps. Figure 4.3 shows patterns in TCP timestamps from about six months during which we ran a full packet capture on the web server, in addition to application-layer logging.

Wilde, and Winter and Lindskog, had found that random "garbage" probes were sent immediately after the client activity that triggered them, while Tor probes were batched and sent only every 15 minutes. The Tor probing behavior had changed by 2015, so that Tor probes were also sent immediately.

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

4.4 Fingerprinting the probers

A potential countermeasure against active probing is to have each proxy, when it receives a connection, somehow decide whether the connection come from a legitimate client or a prober, Of course, the right way to distinguish legitimate clients is with proper cryptographic authentication, whether at the transport layer (like BridgeSPA [130]) or at the application layer (like ScrambleSuit, obfs4, and Shadowsocks). Failing that, one might hope to distinguish probers by their fingerprints, idiosyncrasies in their implementation that make them stand out from legitimate clients. In the case of the Great Firewall, source IP address alone does not suffice because—apart from the special address 202.108.181.70—the probers' source addresses come from many networks, including those where we might expect legitimate clients to reside. There are, however, 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 [35 §4.1], waited for a response, then closed the connection. The VERSIONS cell corresponded to a "v2" Tor handshake that had been superseded since 2011 (though one that was 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 lighter-weight custom implementation.

The obfs2 probes were conformant with the protocol 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 another 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 therefore share some state or a pseudorandom number generator.

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

The SoftEther probes seemed to have been based on an earlier version of the official SoftEther probe than was current at the time, lacking an HTTP Host header. They also differed from the official client in that they were 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 Chromium; however the rest of the header, and the TLS fingerprint are inconsistent with Chromium.

Chapter 5

Time delays in censors' reactions

here be dragons

I am interested in understanding censors at a deeper level. To that end, I am working on a project to measure how long censors take to react to sudden changes in circumvention. So far, our technique has been to monitor the reachability of newly added Tor Browser bridges, to see how long after they are introduced they get blocked. Portions of this work have already appeared in the 2016 research paper "Censors' Delay in Blocking Circumvention Proxies" [66], which I coauthored with Lynn Tsai. We discovered some interesting, previously undocumented behaviors of the Great Firewall of China. While the firewall, through active probing, is able to detect some bridges dynamically within seconds or minutes, it lags in detecting Tor Browser's newly added bridges, taking days or weeks to block them. It seems that bridges are first blocked only at certain times of day, perhaps reflecting an automated batch operation.

I am now continuing to work on this project along with Lynn Tsai and Qi Zhong. We plan to run targeted experiments to find out more about how censors extract bridge addresses from public information, for example, by adding bridges with different attributes and seeing whether they are blocked differently. Our first experiment used measurement sites only in China and Iran, but we hope to expand to many more countries by collaborating with measurement platforms such as OONI [67] and ICLab [79]. We hope to solicit other kinds of censor delays from other circumvention projects, in order to build a more all-encompassing picture of censors' priorities with respect to circumvention.

Chapter 6

Domain fronting

Here be dragons

My most influential contribution to the world of circumvention is my research on domain fronting. While the basic idea is not mine, the research I led and the code I wrote helped domain fronting become the ubiquitous tool it is today.

Three places visible to the censor: * DNS request * SNI * Server certificate And one place not visible to the censor: * Host header

6.1 Related work on domain fronting

[86] Bryce Boe GoAgent flashproxy

Domain fronting assumes a rather strong censor model, essentially equivalent to the state of the art of national censors at the time of its popularization. That is, a censor that can block IP addresses and domain names, that can filter plaintext HTTP, can fingerprint protocol implementations. The main censor capabilities not provided for are probabilistic classification by traffic flow characteristics, and high-collateral-damage blocking of HTTPS on important web servers. What I find most intellectually compelling about domain fronting research is that is finally begins to transcend the "cat-and-mouse" paradigm that has plagued thinking around circumvention, and to put blocking resistance on a scientific basis. By this I mean that one can state assumptions, and consequences that hold as long as the assumptions are true. For example, we do not make claims such as "domain fronting is unblockable"; rather, we may state hypotheses and consequents: "if fronting through a domain with sufficient

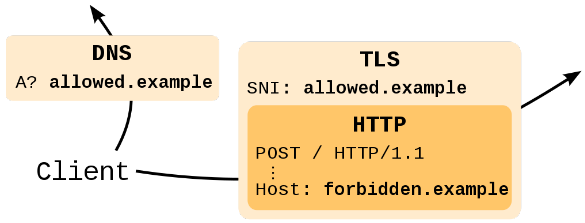

Figure 6.1: Domain fronting uses different names at different network layers.

collateral damage, such that the censor is unwilling to block it, and if the censor does not find some side channel that distinguishes fronted from non-fronted traffic, then the communication will be unblocked." This kind of thinking, that of weighing censors' *costs* and *capabilities*, underlies my thinking about threat modeling.

Like flash proxy, domain fronting is primarily targeted at the problem of address blocking (though it is effective against content blocking and active probing as well). The core idea is the use of different domain names at different layers of communication. The "outside" layers, those visible to the censor, contain an innocuous "front" domain name, ideally one that is hard to block because of the value of the services behind it. The "inside" layer, invisible to the censor under encryption, contains the true, presumably censored, destination. An intermediate server, whose name is the front domain name, removes the outer layer of encryption and forwards the information to the covert destination. There are a number of important services that support domain fronting, mainly cloud providers and content delivery networks. On top of this basic machinery, it is relatively easy to build a general-purpose covert bidirectional communications channel, one that can even be made reasonably efficient.

I wrote and continue to maintain the code of meek, a circumvention transport for Tor based on domain fronting. It first appeared in Tor Browser in October 2014, and continues operation to the present. My code has been forked and incorporated by other circumvention projects, notably including Psiphon and Lantern, with whom I continue to collaborate. Today, meek is Tor's second-most-used transport, carrying around 10 terabytes of user traffic each month.

Köpsell and Hillig were ahead of the game when in 2004 they posed a hypothetical situation [86 §5.2]: "Imagine that all web pages of the United States are only retrievable (from abroad) by sending encrypted request 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." The situation they describe—one server hosting many sites, encrypted and indistinguishably—is not far off from what exists today with CDNs and HTTPS. Domain fronting removes the last remaining easy distinguisher, the domain name that appears in the clear.

Domain fronting appeared in the 2015 research paper "Blocking-resistant communication through domain fronting" [65], which I coauthored with Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson.

CloudTransport [12],

6.2 An unvarnished history of meek deployment

- First release of Orbot that had meek?
- Funding/grant timespans
- cost table
- origin of the name
- "Seeing Through Network-Protocol Obfuscation" [139] October 2015

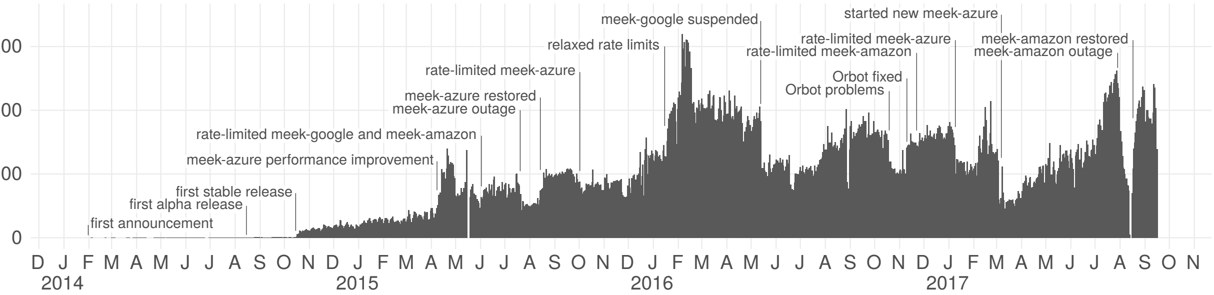

Figure 6.2: Estimated mean number of concurrent users of the meek pluggable transport, with selected events.

- “Towards Measuring Unobservability in Anonymous Communication Systems” [132] October 2015
- “Research and Realization of Tor Anonymous Communication Identification Method Based on Meek”? 2016 http://cdmd.cnki.com.cn/Article/CDMD-10004-1016120870.htm

Fielding a circumvention and keeping it running is full of unexpected challenges. At the time of the publication of the domain fronting paper [65] in 2015, meek had been deployed only a year and a half. Here I will recount the entire history of the deployment project, from inception to the present, a period of over three years. I have been the main developer and project leader of meek over its entire existence. I hope to share the benefit of my experience by commentating the history with surprises and lessons learned. Figure 6.2 shows the estimated concurrent number of users of meek over its entire existence. The counts come from Tor Metrics [91].

2013: Precursors; prototypes

The prehistory of meek begins in 2013 with flash proxy. Flash proxy clients need a secure way to register their address to a central facilitator, in order that flash proxies can connect back to them. Initially we had only two means of registration: flashproxy-reg-http, sending client registrations directly over HTTP; and flashproxy-reg-email, sending client registrations to a special 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 HTTP-rewriting proxy on App Engine. I eventually realized that the same trick would work for flash proxy rendezvous. I proposed a design [11] in May 2013 and within a month Arlo Breault had written flashproxy-reg-appspot, which worked just like flashproxy-reg-http, but fronted through www.google.com rather than contacting the registration server directly. The fronting-based registration became flash proxy's preferred method, being faster and simpler than the email-based one.

The development into a full-fledged bidirectional transport seems slow, in retrospect. All the pieces were there; it was only a matter of putting them together. I did not appreciate the potential of domain fronting when I saw it for the first time. Even after the introduction of flashproxy-reg-appspot, months passed before the beginning of meek. The whole idea behind flash proxy registration was that the registration channel could be of low quality—unidirectional, low-bandwidth, and high-latency—because it was only used to bootstrap into a more capable channel (WebSocket). Email fits well into this model: not good for a general-purpose channel, just good enough for rendezvous. The fronting-based HTTP channel, however, was much more capable, bidirectional with 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 during this time that I first became aware of GoAgent through the "Collateral Freedom" report of Robinson et al. [123]. According to the report, GoAgent, which used a less secure form of domain fronting than what meek would have, 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 [51] which would become meek.

I lost time in 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 the 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 working on a version that strictly serialized request–response pairs, which architecture meek still uses today.

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" [46]. (If the development time seems short, it's only because months of prototypes and false starts.) In the post, I linked to the source code, described the protocol, and explained how to try it, using an App Engine instance I had 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 do the same forwarding as 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 worrying over how to pay for it), and deploying on platforms other than App Engine (with Tor we never used Cloudflare specifically, but did others). 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 bridge. 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," was the kernel of our later paper on domain fronting.

I began the process of getting meek integrated into Tor Browser in February 2014 [61]. 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 [22]) 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 [13]. 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 [53]. Even though our specific application that day did not use domain fronting, the early contact with other circumvention developers was valuable.

June 2014 brought a surprise: the Great Firewall of China blocked all Google services [2, 69]. It would be hubris to think that it was in response to the nascent deployment of meek on App Engine; a more likely cause was Google's decision to start using HTTPS for web searches, which would foil URL keyword 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. At least, we now needed fronts other than Google in order to have any claim of effective circumvention in China. For that reason, 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. I presented domain fronting there. (By this time I had started using the term "domain fronting," realizing that what I had been working on needed a specific name. I tried to separate the idea "domain fronting" from the implementation "meek," but the terms have sometimes gotten confused in discourse.) 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: Percy Wegmann from Lantern and Rod Hynes from Psiphon would later be among my coauthors on the paper on domain fronting [65].

Chang, Vern, and I submitted a paper on domain fronting to the Network and Distributed System Security Symposium (NDSS) in August 2014, whence it was rejected.

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 [17]. 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 [52].

The release and blog post had a positive effect on the number of users, however the absolute numbers are uncertain, because of a configuration error I had made on 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 were spuriously correlated [56]. 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 ($0.12/GB, with one GB free each day) was less than $1.00 per month for the first seven months of 2014 [101 §Costs]. In August, the cost started to be nonzero every day, and would continue to rise from there.

Tor Browser 4.0 [116] 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: the estimate 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 meek-google and meek-azure, individually showed the same increase.) It was a lesson in user behavior: although there had been a working implementation in the alpha release for two months already, evidently a large number of users did not know of it or chose not to try it. 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.

We submitted a revised version of the domain fronting [65], 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 [37] described domain fronting and "collateral freedom" in general, depicting cloud service providers as being caught in the crossfire between censors and circumventors. The journalists had contacted me but I declined to be interviewed. The CEO of CloudFlare, through whose service Lantern had been fronting, said that recently they had altered their systems to prevent domain fronting by enforcing a match between SNI and Host header [118]. GreatFire, an anticensorship organization that had also been mentioned, shortly thereafter experienced a new type of denial-of-service attack [129], caused by a Chinese network attack system later called the "Great Cannon" [96]. They blamed the attack on the attention brought by the news article.

Since initial 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 code I had written to run on Azure, the code that receives domain-fronted HTTP requests and forwards them to the meek bridge, was not reusing TCP connections. For every outgoing request, the Azure code was doing a fresh TCP and TLS handshake—causing a bottleneck at the CPU of the bridge, coping with all the incoming TLS. When I fixed the Azure code to reuse connections [47], the number of users (overall, not only for Azure) had a sudden jump, reaching 6,000 in less than a week. 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 the February 2015, the monthly cost for App Engine alone began to exceed that amount [101 §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, but when the grant expired, we rate-limited the Azure bridge as well. The 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, adding a paragraph that seemed to prohibit running a proxy service [70]:

> *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 an uncomfortable time: we seemed to have the support of Google, but the terms of service said otherwise. I contacted Google and asked 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.

By this time we had not received any reports of any type of blocking of domain fronting. We did, however, suffer a few accidental outages (which look just like blocking, from a user's point of view). Between July 20 and August 14, an account transition error left the Azure configuration broken [55]. 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-fronted 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 would refuse to connect to it [63]. 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.

One of the benefits of building a circumvention system for Tor is the 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' data collection. 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 is some cloud server, the intermediary through which the domain-fronted traffic passes. So the total 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 in the U.S. bucket. By the end of 2015, meek users were a large enough fraction (about 20%) of all bridge users, that they were really starting to skew the overall per-country counts. I wrote a patch 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. (The blocking attempts were in turn rumored to be caused by Facebook's integration of Tor into their mobile application.) I had the bridge operators raise the rate 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 becoming Tor's most-used pluggable transport, before settling in 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 domain fronting, but also thousands of other names, became inaccessible. The block lasted about four days. On February 2, the server changed its IP address, incrementing the final octet from .200 to .201, causing it to become unblocked. I am aware of no similar incidents before or since.

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 was that it had to do with the changes to the terms of service that had happened the previous year—but the true cause was unexpected. I tried repeatedly to contact Google and learn the nature of the "general" violation, but was stonewalled. None of my inquiries received so much as an acknowlegement. It as not until June 18 that I got some insight as to what happened, through an unofficial channel. Some botnet had apparently been misusing meek for command and control purposes; and its operators hadn't even bothered to set up their own App Engine project. They were using the service that 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 botnet, with the already uncertain standing with regard to 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. Matthew Dunwoody presented observations to that effect in a FireEye blog post [38] in March 2017. He and Nick Carr had presented those findings at DerbyCon in

September 2016 [39], but I was not aware of them until the blog post. Malware would install a backdoor that operated over a Tor onion service, and used meek for camouflage.

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 [81]. 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 [49]. 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 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 spent the time to implement browser-based TLS camouflage. And there was no error in the camouflage: even an ordinary Firefox 38 (the base for Tor Browser, and what meek camouflaged itself as) was blocked by the firewall when accessing one of the three front domains. However, Firefox 38 was by that time a year old. I found a source saying that it made up only 0.38% of desktop browsers, compared to 10.69% for the then-latest Firefox 45 [49]. My guess is that the firewall makers considered the small amount of collateral blocking of Firefox 38 users to be acceptable.

In July I received a report of similar behavior by a FortiGuard firewall [50] from Tor user Kanwaljeet Singh Channey. The situation was virtually the same: 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 only two, omitting www.google.com. (This meant that meek-google would have worked, had it not been deactivated back in May.) As in the Cyberoam 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 of Tor users in that country in June 2016 [64]. 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 meek blocking that I am aware of.

Starting in July 2016, there was a months-long increase in the number of meek users reported from Brazil [133]. 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 dropped suddenly, from 1,500 almost to zero, on March 3, 2017, which happened also to be the day that meek-azure was shut down pending a migration to new infrastructure. The count would remain low until rising again in June 2017.

In September 2016, I began mentoring Katherine Li in making her program GAEuploader [89], which aims 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 line ready to be pasted into Tor Browser or Orbot. We hoped also that the code would be useful to other projects, like XX-Net [155], that provide documentation on the complicated process of uploading code to App Engine. GAEuploader had a beta release in January 2017 [88]; however the effect on the number of users was not substantial.

Between October 19 and November 10, 2016, the number of meek users decreased globally by about a third [62]. Initially I suspected a censorship event, but the other details didn't add up: the numbers were depressed 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 [58]. Once a fixed release was available, user numbers recovered. An unanticipated effect of this occurrence was that we learned that a majority of meek users were using Orbot rather than Tor Browser.

2017: Long-term support

In January 2017, the 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 [54]. Shortly thereafter, Team Cymru offered to stand 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 release of Tor Browser was 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 completely non-functional for users. 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 in this case it wasn't 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 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 [59]. They claimed support for "Psiphon CDN (Meek mode)" going back to January 2015, and for "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, similarly to how it had between July 2016 and March 2017. Just as before, we did not find an explanation for the increase.

Between July 29 and August 17, meek-amazon had another outage due to an expired TLS certificate.

Chapter 7

Snowflake

here be dragons

Flash proxy revisited

WebRTC fingerprinting

Engineering challenges

I am working on a new circumvention system, a transport for Tor called Snowflake. Snowflake is the successor to flash proxy. It keeps the basic idea of in-browser proxies while fixing the usability problems that hampered the adoption of flash proxy. My main collaborators in this project are Arlo Breault, Serene Han, Mia Gil Epner, and Hooman Mohajeri.

The key difference between flash proxy and Snowflake is the basic communications protocol between client and browser proxy. Flash proxy used the TCP-based WebSocket protocol, which required users to configure their personal firewall to allow incoming connections. Snowflake instead uses WebRTC, a UDP-based protocol that enables peer-to-peer connections without manual configuration. The most similar existing system is uProxy [137], which in one of its operating modes uses WebRTC to connect through a friend's computer. Snowflake differs because it does not require prior coordination with a friend before connecting. Instead,

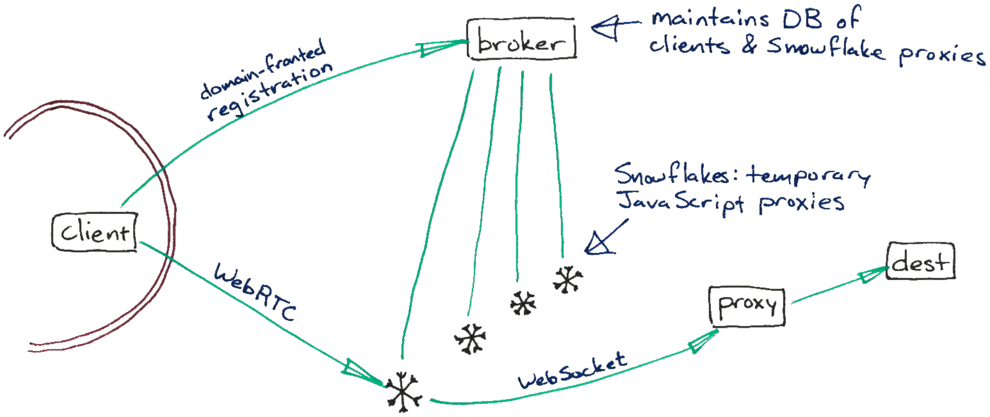

Figure 7.1: Diagram of Snowflake.

it pulls its proxies from a pool of web users who are running the Snowflake code. Beyond the changed protocol, we hope to build in performance and efficiency improvements.

Snowflake will afford interesting research opportunities. One, of course, is the design of the system itself—no circumvention system of its nature has previously been deployed at a large scale. Another opportunity is observing how censors react to a new challenge.

Most of the available documentation on Snowflake is linked from the project's wiki page [131]. Mia Gil Epner and I wrote a technical report on the fingerprinting hazards of WebRTC [57].

7.0.1 Flash proxy

I began working on censorship circumvention with flash proxy in 2011. Flash proxy is targeted at the difficult problem of proxy address blocking: it is designed against a censor model in which the censor can block any IP address it chooses, but only on a relatively slow timeline of several hours.

Flash proxy works by running tiny JavaScript proxies in ordinary users' web browsers. The mini-proxies serve as temporary stepping stones to a full-fledged proxy, such as a Tor relay. The idea is that the flash proxies are too numerous, diverse, and quickly changing to block effectively. A censored user 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.

The flash proxy system was designed under interesting constraints imposed by being partly implemented in JavaScript in the browser. The proxies sent and received data using the WebSocket protocol, which allows for socket-like persistent TCP connections in browsers, but with a catch: the browser can only make outgoing connections, not receive incoming ones as a traditional proxy would. The censored client must somehow inform the system of its own public address, and then the proxy connects *back* to the client. This architectural constraint was probably the biggest impediment to the usability of flash proxy, because it required users to configure their local router to permit incoming connections. (Removing this impediment is the main reason for the development of Snowflake, described later.) Flash proxy does not itself try to obfuscate patterns in the underlying traffic; it only provides address diversity.

For the initial "rendezvous" step in which a client advertises its address and a request for proxy service, flash proxy uses a neat idea: a low-capacity, but highly covert channel bootstraps the high-capacity, general-purpose WebSocket channel. For example, we implemented an automated email-based rendezvous, in which the client would send its address in an encrypted email to a special address. While it is difficult to build a useful low-latency bidirectional channel on top of email, email is difficult to block and it is only needed once, at the beginning of a session. We later replaced the email-based rendezvous with one based on domain fronting, which would later inspire meek, described below.

I was the leader of the flash proxy project and the main developer of its code. Flash proxy was among the first circumvention systems built for Tor—only obfs2 is older. It was first deployed in Tor Browser in January 2013, and was later retired in January 2016 after it ceased to see appreciable use. Its spirit lives on in Snowflake, now under development.

Flash proxy appeared in the 2012 research paper "Evading Censorship with Browser-Based Proxies" [60], which I coauthored with Nate Hardison, Jonathan Ellithorpe, Emily Stark, Roger Dingledine, Phil Porras, and Dan Boneh.

Appendix A

Summary of censorship measurement studies

Here I survey past measurement studies which have helped to build models about censor behavior in general. The objects of the survey are based on those in an evaluation study done by me and others in 2016 [135 §IV.A].

One of the earliest technical studies of censorship occurred not in some illiberal place, but in the German state of North Rhein-Westphalia. In 2003, Dornseif [36] tested ISPs' implementation of a controversial legal order to block two Nazi web sites. 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*, simply returning (or injecting) false responses to DNS requests for the domain names of 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.

Clayton [18] in 2006 studied a "hybrid" blocking system, called "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 pedophile web sites compiled by a third party. The author identifies ways to circumvent or attack such a system: use a proxy, use source routing to evade the blocking router, obfuscate requested URLs, use an alternate IP address or port, return false DNS results to put third parties on the "bad" list. They demonstrate that the two-level nature of the blocking system unintentionally makes it an oracle that can reveal the IP addresses of sites in the secret blocking list.

[26]

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 [111] studied the problem from many perspectives, political, technical, and legal. They translated and interpreted Chinese laws relating to the Internet, which provide strong legal justifications for filtering. The laws regulate both Internet users and service providers, including cybercafes. They prohibit the transfer of information that is indecent, subversive, false, criminal, or that reveals state secrets. The OpenNet Initiative 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 sites (domains) were blocked; in others, only specific pages within a larger site were blocked. In a small number of cases, sites were accessible by IP address but not by domain name. There were cases of overblocking: apparently inadvertently blocked sites that simply shared an IP address or URL keyword with an intentionally blocked site. On seeing a prohibited keyword, the firewall blocked connections by injecting a TCP RST packet to tear down the connection, 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 index blocked sites (perhaps having a special exemption from the general firewall policy), but do not return them in search results. The firewall blocks access searches for certain keywords on Google as well as the Google Cache—but the latter could be worked around by tweaking the format of the URL. Censorship of blogs comprised keyword blocking by domestic blogging services, and blocking of external domains such as blogspot.com. Email filtering is done by the email providers themselves, not by an independent network firewall. Email providers seem to implement their filtering rules independently and inconsistently: messages were blocked by some providers and not others.

In 2006, Clayton, Murdoch, and Watson [19] 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 (e.g., “falun”) 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 of this paper bring 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 choose the architecture of a network monitor or intrusion detection system, one that can passively monitor and inject packets, but cannot delay or drop them.

A nearly contemporary study by Wolfgarten [151] reproduced many of the results of Clayton, Murdoch, and Watson. Using a rented server in China, the author found cases of DNS tampering, search engine filtering, and RST injection caused by keyword sniffing. Not much later, in 2007, Lowe, Winters, and Marcus [93] 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 TTLs, the authors found that the false responses were injected by an intermediate router: the authentic response would be received as well, only later. A more comprehensive survey [7] of DNS tampering and injection occurred in 2014, giving remarkable insight into the internal structure of the censorship machines. DNS injection happens only at border routers. IP ID and TTL analysis show that each node is a cluster of several hundred processes that collectively inject censored responses. They found 174 bogus IP addresses, more than previously documented. They extracted a blacklist of about 15,000 keywords.

[152]

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. [21] 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 [113] 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. [154] continued the theme of keyword filtering in 2011, with the goal of discovering where filters are located at the IP and AS levels. Most filtering is done at border networks (autonomous systems with at least one non-Chinese peer). In their measurements, the firewall was fully stateful: blocking was never triggered by an HTTP request outside an established TCP connection. Much filtering occurs at smaller regional providers, rather than on the network backbone.

Winter and Lindskog [149] 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 relays. Using private Tor relays in Singapore, Sweden, and Russia, they provoked active probes by simulating Tor connections, collecting 3295 firewall scans over 17 days. Over half the scan came from a single IP address in China; the remainder seemingly came from ISP pools. Active probing is initiated every 15 minutes and each burst lasts for about 10 minutes.

Sfakianakis et al. [126] built CensMon, a system for testing web censorship using PlanetLab nodes as distributed measurement points. They ran the system for 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 reports the mechanism of blocking. Across all nodes, it was 18.2% DNS tampering, 33.3% IP address blocking, and 48.5% HTTP keyword filtering. The system was not run on a continuing basis. Verkamp and Gupta [138] did a separate study in 11 countries, using a combination of PlanetLab nodes and the computers of volunteers. Censorship techniques vary across countries; for example, some show overt block pages and others do not. China was the only stateful censor of the 11.

PlanetLab is a system that was not originally designed for censorship measurement, that was later adapted for that purpose. Another recent example is RIPE Atlas, a globally distributed Internet measurement network consisting of physical probes hosted by volunteers, Atlas allows 4 types of measurements: ping, traceroute, DNS resolution, and X.509 certificate fetching. Anderson et al. [4] used Atlas 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. 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.

Most research on censors has focused on the blocking of specific web sites and HTTP keywords. A few studies have looked at less discriminating forms of censorship: outright shutdowns and throttling without fully blocking. Dainotti et al. [25] 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: BGP data, a large network telescope, and active traceroutes. During outages, there was a drop in scanning traffic (mainly from the Conficker botnet) to their telescope. By comparing these different measurements, they showed that the shutdown in Libya was accomplished in more that one way, both by altering network routes and by firewalls dropping packets. Anderson [3] 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. The author argues that a hallmark of throttling is a decrease in network throughput without an accompanying increase in latency and packet loss, distinguishing throttling from ordinary network congestion. Academic institutions were affected by throttling, but less so than other networks. Aryan et al. [8] 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 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, thereby forcing TCP's usual recovery.

Khattak et al. [85] evaluated the Great Firewall from the perspective that it works like an intrusion detection system or network monitor, and applied existing technique for evading a monitor the 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 is stateful, but only in the client-to-server direction. The firewall is vulnerable to a variety of TCP- and HTTP-based evasion techniques, such as overlapping fragments, TTL-limited packets, and URL encodings.

Nabi [105] investigated web censorship in Pakistan in 2013, using a publicly known list of banned web sites. They tested on 5 different networks in Pakistan. Over half of the sites on the list were blocked by DNS tampering; less than 2% were additionally blocked by HTTP filtering (an injected redirection before April 2013, or a static block page after that). They conducted a small survey to find the most commonly used circumvention methods in Pakistan. The most used method was public VPNs, at 45% of respondents.

Ensafi et al. [44] 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 lasting in bursts of hours.

In 2015, Marczak et al. [96] 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 in China. The new attack tool is noteworthy because it demonstrated previously unseen in-path behavior, such as packet dropping.

Not every censor is China, with its sophisticated homegrown firewall. A major aspect of censor modeling is that many censors use commercial firewall hardware. A case in point is the analysis by Chaabane et al. [15] of 600 GB of leaked logs from Blue Coat proxies 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, domain name blocking, and HTTP request keyword blocking, and also of users circumventing censorship by downloading circumvention software or using the Google cache. 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", "hotspotshield", "israel", and "ultrasurf" (resulting in collateral damage to the Google Toolbar and Facebook Like button because they have "proxy" in HTTP requests). Tor was only lightly censored—only one of several proxies blocked it, and only sporadically.

[71] and other OONI.

Analyzing Internet Censorship in Pakistan[1]

Bibliography

[1] 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. 47).

[2] 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. 34).

[3] 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. 46).

[4] 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. 45).

[5] 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 p. 15).

[6] 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).

[7] 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 pp. 19, 44).

[8] 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 p. 46).

[9] 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. 16).

[10] 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. 23, 24).

[11] Arlo Breault, David Fifield, and George Kadianakis. *Registration over App Engine*. May 2013. https://bugs.torproject.org/8860 (cit. on p. 32).

[12] 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, 31).

[13] Willow Brugh. *San Francisco Hackathon/DiscoTech (+ RightsCon + Responsible Data Forum)*. Mar. 2014. http://codesign.mit.edu/2014/03/sfdiscotech/ (cit. on p. 34).

[14] 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. 20).

[15] 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. 47).

[16] The Citizen Lab. *Psiphon*. Oct. 2006. https://web.archive.org/web/20061026081356/http://psiphon.civisec.org/ (cit. on p. 16).

[17] 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, 34).

[18] 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. 43).

[19] 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. 8, 14, 44).

[20] 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. http://ensr.oii.ox.ac.uk/wp-content/uploads/2015/07/Forgive-Us-Our-SYNs-Technical-and-Ethical-Considerations-for-Measuring-Internet-Censorship.pdf (cit. on p. 20).

[21] 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, 18, 45).

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

[23] 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).

[24] 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. 15).

[25] Alberto Dainotti, Claudio Squarcella, Emile Aben, Kimberly C. Claffy, Marco Chiesa, Michele Russo, and Antonio Pescapé. “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. 46).

[26] Ronald Deibert, John Palfrey, Rafal Rohozinski, and Jonathan Zittrain, eds. *Access denied: the practice and policy of global Internet filtering*. Cambridge, Mass: MIT Press, 2008. ISBN: 978-0-262-54196-1. http://access.opennet.net/?page_id=61 (cit. on p. 43).

[27] 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 p. 23).

[28] 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. 11, 23, 24).

[29] 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. 13).

[30] 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. 13).

[31] 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. 13, 21).

[32] 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. 23, 24).

[33] 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. 23, 24).

[34] 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. 12, 13, 16, 21).

[35] Roger Dingledine and Nick Mathewson. *Tor Protocol Specification*. Sept. 2017. https://spec.torproject.org/tor-spec (cit. on p. 28).

[36] 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. 43).

[37] Eva Dou and Alistair Barr. *U.S. Cloud Providers Face Backlash From China's Censors.* Wall Street Journal. Mar. 2015. https://www.wsj.com/articles/u-s-cloud-providers-face-backlash-from-chinas-censors-1426541126 (cit. on p. 35).

[38] 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. 37).

[39] 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. 38).

[40] 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. 13, 22).

[41] 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).

[42] Kevin P. Dyer, Scott E. Coull, and Thomas Shrimpton. "Marionette: A Programmable Network-Traffic Obfuscation System". In: *USENIX Security Symposium.* USENIX, 2015. https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-dyer.pdf (cit. on p. 10).

[43] 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. 11, 23–26).

[44] 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 p. 46).

[45] 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).

[46] 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. 33).

[47] 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. 35).

[48] 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. 23, 24).

[49] 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. 38).

[50] 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. 38).

[51] 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. 33).

[52] 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. 34).

[53] 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. 34).

[54] 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. 39).

[55] 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. 36).

[56] 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. 35).

[57] David Fifield and Mia Gil Epner. *Fingerprintability of WebRTC*. Tech. rep. May 2016. https://arxiv.org/abs/1605.08805v1 (cit. on p. 41).

[58] 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. 39).

[59] 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. 39).

[60] David Fifield, Nate Hardison, Jonathan Ellithorpe, Emily Stark, Roger Dingledine, Phil 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, 41).

[61] David Fifield, George Kadianakis, Georg Koppen, and Mark Smith. *Make bundles featuring meek*. Feb. 2014. https://bugs.torproject.org/10935 (cit. on p. 34).

[62] 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. 39).

[63] David Fifield, Georg Koppen, and Klaus Layer. *Update the meek-amazon fingerprint to B9E7141C594AF25699E0079C1F0146F409495296*. Oct. 2015. https://bugs.torproject.org/17473 (cit. on p. 36).

[64] 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 p. 38).

[65] 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. 14, 31, 32, 34, 35).

[66] 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 pp. 8, 29).

[67] 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 pp. 18, 19, 29).

[68] 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. 10).

[69] Google. *Google Transparency Report: China, All Products, May 31, 2014–Present*. July 2014. https://www.google.com/transparencyreport/traffic/disruptions/124/ (cit. on p. 34).

[70] Google Cloud Platform. *Service Specific Terms*. Mar. 2015. https://web.archive.org/web/20150326000133/https://cloud.google.com/terms/service-terms (cit. on p. 36).

[71] Arthur Gwagwa. *A study of Internet-based information controls in Rwanda, with a particular focus on the period around the 4 August 2017 General Elections*. Oct. 2017. https://www.opentech.fund/sites/default/files/attachments/a_study_of_internet-based_information_controls_in_rwanda-arthur_gwagwa_final.pdf (cit. on p. 47).

[72] Bennett Haselton. *Circumventor*. Peacefire. http://peacefire.org/circumventor/ (cit. on p. 16).

[73] Bennett Haselton. *Peacefire Censorware Pages*. Peacefire. http://www.peacefire.org/censorware/ (cit. on p. 16).

[74] 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. 23, 25).

[75] 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, 10).

[76] 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 p. 8).

[77] 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).

[78] 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. 14).

[79] *ICLab*. https://iclab.org/ (cit. on pp. 19, 29).

[80] 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. 20).

[81] 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. 38).

[82] 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. 11).

[83] 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. 11).

[84] 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).

[85] 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, 46).

[86] 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, 30, 31).

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

[88] Katherine Li. *GAEuploader.* tor-dev mailing list. Jan. 2017. https://lists.torproject.org/pipermail/tor-dev/2017-January/011812.html (cit. on p. 39).

[89] Katherine Li. *GAEuploader.* https://github.com/katherinelitor/GAEuploader (cit. on p. 39).

[90] Patrick Lincoln, Ian Mason, Phillip Porras, Vinod Yegneswaran, Zachary Weinberg, Jeroen Massar, William Simpson, Paul Vixie, and Dan Boneh. “Bootstrapping Communications into an Anti-Censorship System”. In: *Free and Open Communications on the Internet.* USENIX, 2012. https://www.usenix.org/system/files/conference/foci12/foci12-final7.pdf (cit. on p. 12).

[91] Karsten Loesing. *Counting daily bridge users.* Tech. rep. 2012-10-001. The Tor Project, Oct. 2012. https://research.torproject.org/techreports/counting-daily-bridge-users-2012-10-24.pdf (cit. on p. 32).

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

[93] 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. 44).

[94] Max Lv and Rio. *AEAD Ciphers.* https://shadowsocks.org/en/spec/AEAD-Ciphers.html (cit. on p. 22).

[95] Marek Majkowski. *Fun with The Great Firewall.* July 2013. https://idea.popcount.org/2013-07-11-fun-with-the-great-firewall/ (cit. on pp. 23, 24).

[96] 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. 35, 47).

[97] James Marshall. *CGIProxy.* https://jmarshall.com/tools/cgiproxy/ (cit. on p. 16).

[98] 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. 16).

[99] 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, 22).

[100] 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. 21).

[101] *meek.* Tor Bug Tracker & Wiki. https://trac.torproject.org/projects/tor/wiki/doc/meek (cit. on pp. 35, 36).

[102] 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. 16).

[103] 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).

[104] Rich Morin. "The Limits of Control". In: *Unix Review Magazine* (June 1996). http://cfcl.com/rdm/Pubs/tin/P/199606.shtml (cit. on p. 16).

[105] 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. 46).

[106] Abhinav Narain, Nick Feamster, and Alex C. Snoeren. "Deniable Liaisons". In: *Computer and Communications Security.* ACM, 2014. https://cseweb.ucsd.edu/~snoeren/papers/denali-ccs14.pdf (cit. on pp. 8, 14).

[107] Milad Nasr, Sadegh Farhang, Amir Houmansadr, and Jens Grossklags. *Enemy At the Gateways: A Game Theoretic Approach to Proxy Distribution.* Tech. rep. Sept. 2017. https://arxiv.org/abs/1709.04030v1 (cit. on p. 13).

[108] NetFreedom Pioneers. *Toosheh.* https://www.toosheh.org/en.html (cit. on p. 15).

[109] Leif Nixon. *Some observations on the Great Firewall of China.* Nov. 2011. https://www.nsc.liu.se/~nixon/sshprobes.html (cit. on p. 23).

[110] 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, 13, 25).

[111] OpenNet Initiative. *Internet Filtering in China in 2004-2005: A Country Study.* https://opennet.net/studies/china (cit. on p. 43).

[112] Overthrow CPC. 关于 *ss/ssr.* 编程随想的博客. Oct. 2017. https://program-think.blogspot.com/2017/10/gfw-news.html?comment=1508314948860 (cit. on p. 25).

[113] 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, 45).

[114] 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. 15).

[115] Mike Perry. *Tor Browser 3.6 is released.* The Tor Blog. Apr. 2014. https://blog.torproject.org/tor-browser-36-released (cit. on p. 11).

[116] Mike Perry. *Tor Browser 4.0 is released*. The Tor Blog. Oct. 2014. https://blog.torproject.org/tor-browser-40-released (cit. on pp. 23, 24, 35).

[117] Mike Perry. *Tor Browser 4.5 is released*. The Tor Blog. Apr. 2015. https://blog.torproject.org/tor-browser-45-released (cit. on pp. 23, 24).

[118] Matthew Prince. Hacker News. Mar. 2015. https://news.ycombinator.com/item?id=9234367 (cit. on p. 35).

[119] 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. 22–24).

[120] 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. 15).

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

[122] 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. 20).

[123] 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. 33).

[124] SafeWeb. *TriangleBoy Whitepaper*. http://www.webrant.com/safeweb_site/html/www/tboy_whitepaper.html (cit. on p. 14).

[125] 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. 14).

[126] 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 pp. 18, 45).

[127] *Shadowsocks*. https://shadowsocks.org/en/ (cit. on p. 11).

[128] Thomas Sladek and Eduard Bröse. *Market Survey: Detection & Filtering Solutions to Identify File Transfer of Copyright Protected Content for Warner Bros. and movielabs*. Tech. rep. EANTC AG, Mar. 2011. https://wikileaks.org/sony/docs/05/docs/Anti-Piracy/CDSA/EANTC-Survey-1.5-unsecured.pdf (cit. on p. 9).

[129] Charlie Smith. *We are under attack*. GreatFire. Mar. 2015. https://en.greatfire.org/blog/2015/mar/we-are-under-attack (cit. on p. 35).

[130] 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. 28).

[131] *Snowflake*. Tor Bug Tracker & Wiki. https://trac.torproject.org/projects/tor/wiki/doc/Snowflake (cit. on pp. 13, 41).

[132] 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 p. 32).

[133] Tor Metrics. *Bridge users by transport from Brazil*. Oct. 2017. https://metrics.torproject.org/userstats-bridge-combined.html?start=2016-06-01&end=2017-10-01&country=br (cit. on p. 38).

[134] The Tor Project. *BridgeDB*. https://bridges.torproject.org/ (cit. on p. 12).

[135] 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, 20, 43).

[136] 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. 22).

[137] *uProxy*. https://www.uproxy.org/ (cit. on pp. 12, 40).

[138] 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. 45).

[139] 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. 10, 31).

[140] 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. 14).

[141] 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 p. 15).

[142] 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).

[143] Tim Wilde. *CN Prober IPs*. Dec. 2011. https://gist.github.com/twilde/4320b75d398f2e1f074d (cit. on p. 24).

[144] Tim Wilde. *Great Firewall Tor Probing Circa 09 DEC 2011*. Jan. 2012. https://gist.github.com/twilde/da3c7a9af01d74cd7de7 (cit. on pp. 23, 24).

[145] 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. 23, 24).

[146] 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. 11).

[147] Philipp Winter. *brdgrd*. 2012. https://github.com/NullHypothesis/brdgrd (cit. on pp. 15, 24).

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

[149] 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, 23, 24, 45).

[150] 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, 22).

[151] 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. 44).

[152] Joss Wright. *Regional Variation in Chinese Internet Filtering*. Tech. rep. University of Oxford, 2012. https://papers.ssrn.com/sol3/Delivery.cfm/SSRN_ID2265775_code1448244.pdf?abstractid=2265775&mirid=3 (cit. on p. 45).

[153] 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, 20).

[154] 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. 45).

[155] *XX-Net*. https://github.com/XX-net/XX-Net (cit. on p. 39).

[156] 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, 22).