Design of a blocking-resistant anonymity system DRAFT

Roger Dingledine and Nick Mathewson

The Free Haven Project
{arma,nickm}@freehaven.net

Abstract. Internet censorship is on the rise as websites around the world are increasingly blocked by government-level firewalls. Although popular anonymizing networks like Tor were originally designed to keep attackers from tracing people's activities, many people are also using them to evade local censorship. But if the censor simply denies access to the Tor network itself, blocked users can no longer benefit from the security Tor offers.
Here we describe a design that builds upon the current Tor network to provide an anonymizing network that resists blocking by government-level attackers.

1 Introduction and Goals

Anonymizing networks like Tor [11] bounce traffic around a network of encrypting relays. Unlike encryption, which hides only *what* is said, these networks also aim to hide who is communicating with whom, which users are using which websites, and similar relations. These systems have a broad range of users, including ordinary citizens who want to avoid being profiled for targeted advertisements, corporations who don't want to reveal information to their competitors, and law enforcement and government intelligence agencies who need to do operations on the Internet without being noticed.

Historical anonymity research has focused on an attacker who monitors the user (call her Alice) and tries to discover her activities, yet lets her reach any piece of the network. In more modern threat models such as Tor's, the adversary is allowed to perform active attacks such as modifying communications to trick Alice into revealing her destination, or intercepting some connections to run a man-in-the-middle attack. But these systems still assume that Alice can eventually reach the anonymizing network.

An increasing number of users are using the Tor software less for its anonymity properties than for its censorship resistance properties—if they use Tor to access Internet sites like Wikipedia and Blogspot, they are no longer affected by local censorship and firewall rules. In fact, an informal user study showed China as the third largest user base for Tor clients, with perhaps ten thousand people accessing the Tor network from China each day.

The current Tor design is easy to block if the attacker controls Alice's connection to the Tor network—by blocking the directory authorities, by blocking all the server IP addresses in the directory, or by filtering based on the signature of the Tor TLS handshake. Here we describe an extended design that builds upon the current Tor network to provide an anonymizing network that resists censorship as well as anonymity-breaking attacks. In section 2 we discuss our threat model—that is, the assumptions we make about our adversary. Section 3 describes the components of the current Tor design and how they can be leveraged for a new blocking-resistant design. Section 4 explains the features and drawbacks of the currently deployed solutions. In sections 5 through 7, we explore the components of our designs in detail. Section 8 considers security implications and Section 9 presents other issues with maintaining connectivity and sustainability for the design. Section 10 speculates about future more complex designs, and finally Section 11 summarizes our next steps and recommendations.

2 Adversary assumptions

To design an effective anti-censorship tool, we need a good model for the goals and resources of the censors we are evading. Otherwise, we risk spending our effort on keeping the adversaries from doing things they have no interest in doing, and thwarting techniques they do not use. The history of blocking-resistance designs is littered with conflicting assumptions about what adversaries to expect and what problems are in the critical path to a solution. Here we describe our best understanding of the current situation around the world.

In the traditional security style, we aim to defeat a strong attacker—if we can defend against this attacker, we inherit protection against weaker attackers as well. After all, we want a general design that will work for citizens of China, Iran, Thailand, and other censored countries; for whistleblowers in firewalled corporate networks; and for people in unanticipated oppressive situations. In fact, by designing with a variety of adversaries in mind, we can take advantage of the fact that adversaries will be in different stages of the arms race at each location, so a server blocked in one locale can still be useful in others.

We assume that the attackers' goals are somewhat complex.

- The attacker would like to restrict the flow of certain kinds of information, particularly when this information is seen as embarrassing to those in power (such as information about rights violations or corruption), or when it enables or encourages others to oppose them effectively (such as information about opposition movements or sites that are used to organize protests).
- As a second-order effect, censors aim to chill citizens' behavior by creating an impression that their online activities are monitored.
- In some cases, censors make a token attempt to block a few sites for obscenity, blasphemy, and so on, but their efforts here are mainly for show. In other cases, they really do try hard to block such content.
- Complete blocking (where nobody at all can ever download censored content) is not a goal. Attackers typically recognize that perfect censorship is not only impossible, but unnecessary: if "undesirable" information is known only to a small few, further censoring efforts can be focused elsewhere.
- Similarly, the censors are not attempting to shut down or block *every* anti-censorship tool—merely the tools that are popular and effective (because these tools impede the censors' information restriction goals) and those tools that are highly visible (thus making the censors look ineffectual to their citizens and their bosses).
- Reprisal against *most* passive consumers of *most* kinds of blocked information is also not a goal, given the broadness of most censorship regimes. This seems borne out by fact.[1]
- Producers and distributors of targeted information are in much greater danger than consumers; the attacker would like to not only block their work, but identify them for reprisal.
- The censors (or their governments) would like to have a working, useful Internet. There are economic, political, and social factors that prevent them from "censoring" the Internet by outlawing it entirely, or by blocking access to all but a tiny list of sites. Nevertheless, the censors *are* willing to block innocuous content (like the bulk of a newspaper's reporting) in order to censor other content distributed through the same channels (like that newspaper's coverage of the censored country).

We assume there are three main technical network attacks in use by censors currently [7]:

[1] So far in places like China, the authorities mainly go after people who publish materials and coordinate organized movements [22]. If they find that a user happens to be reading a site that should be blocked, the typical response is simply to block the site. Of course, even with an encrypted connection, the adversary may be able to distinguish readers from publishers by observing whether Alice is mostly downloading bytes or mostly uploading them—we discuss this issue more in Section 8.2.

- Block a destination or type of traffic by automatically searching for certain strings or patterns in TCP packets. Offending packets can be dropped, or can trigger a response like closing the connection.
- Block a destination by listing its IP address at a firewall or other routing control point.
- Intercept DNS requests and give bogus responses for certain destination hostnames.

We assume the network firewall has limited CPU and memory per connection [7]. Against an adversary who could carefully examine the contents of every packet and correlate the packets in every stream on the network, we would need some stronger mechanism such as steganography, which introduces its own problems [15, 26]. But we make a "weak steganography" assumption here: to remain unblocked, it is necessary to remain unobservable only by computational resources on par with a modern router, firewall, proxy, or IDS.

We assume that while various different regimes can coordinate and share notes, there will be a time lag between one attacker learning how to overcome a facet of our design and other attackers picking it up. (The most common vector of transmission seems to be commercial providers of censorship tools: once a provider adds a feature to meet one country's needs or requests, the feature is available to all of the provider's customers.) Conversely, we assume that insider attacks become a higher risk only after the early stages of network development, once the system has reached a certain level of success and visibility.

We do not assume that government-level attackers are always uniform across the country. For example, users of different ISPs in China experience different censorship policies and mechanisms.

We assume that the attacker may be able to use political and economic resources to secure the cooperation of extraterritorial or multinational corporations and entities in investigating information sources. For example, the censors can threaten the service providers of troublesome blogs with economic reprisals if they do not reveal the authors' identities.

We assume that our users have control over their hardware and software—they don't have any spyware installed, there are no cameras watching their screens, etc. Unfortunately, in many situations these threats are real [28]; yet software-based security systems like ours are poorly equipped to handle a user who is entirely observed and controlled by the adversary. See Section 8.4 for more discussion of what little we can do about this issue.

Similarly, we assume that the user will be able to fetch a genuine version of Tor, rather than one supplied by the adversary; see Section 8.5 for discussion on helping the user confirm that he has a genuine version and that he can connect to the real Tor network.

3 Adapting the current Tor design to anti-censorship

Tor is popular and sees a lot of use—it's the largest anonymity network of its kind, and has attracted more than 800 volunteer-operated routers from around the world. Tor protects each user by routing their traffic through a multiply encrypted "circuit" built of a few randomly selected servers, each of which can remove only a single layer of encryption. Each server sees only the step before it and the step after it in the circuit, and so no single server can learn the connection between a user and her chosen communication partners. In this section, we examine some of the reasons why Tor has become popular, with particular emphasis to how we can take advantage of these properties for a blocking-resistance design.

Tor aims to provide three security properties:

- 1. A local network attacker can't learn, or influence, your destination.
- 2. No single router in the Tor network can link you to your destination.
- 3. The destination, or somebody watching the destination, can't learn your location.

For blocking-resistance, we care most clearly about the first property. But as the arms race progresses, the second property will become important—for example, to discourage an adversary from volunteering a relay in order to learn that Alice is reading or posting to certain websites. The third property helps keep users safe from collaborating websites: consider websites and other Internet services that have been pressured recently into revealing the identity of bloggers or treating clients differently depending on their network location [17].

The Tor design provides other features as well that are not typically present in manual or ad hoc circumvention techniques.

First, Tor has a well-analyzed and well-understood way to distribute information about servers. Tor directory authorities automatically aggregate, test, and publish signed summaries of the available Tor routers. Tor clients can fetch these summaries to learn which routers are available and which routers are suitable for their needs. Directory information is cached throughout the Tor network, so once clients have bootstrapped they never need to interact with the authorities directly. (To tolerate a minority of compromised directory authorities, we use a threshold trust scheme— see Section 8.5 for details.)

Second, the list of directory authorities is not hard-wired. Clients use the default authorities if no others are specified, but it's easy to start a separate (or even overlapping) Tor network just by running a different set of authorities and convincing users to prefer a modified client. For example, we could launch a distinct Tor network inside China; some users could even use an aggregate network made up of both the main network and the China network. (But we should not be too quick to create other Tor networks—part of Tor's anonymity comes from users behaving like other users, and there are many unsolved anonymity questions if different users know about different pieces of the network.)

Third, in addition to automatically learning from the chosen directories which Tor routers are available and working, Tor takes care of building paths through the network and rebuilding them as needed. So the user never has to know how paths are chosen, never has to manually pick working proxies, and so on. More generally, at its core the Tor protocol is simply a tool that can build paths given a set of routers. Tor is quite flexible about how it learns about the routers and how it chooses the paths. Harvard's Blossom project [16] makes this flexibility more concrete: Blossom makes use of Tor not for its security properties but for its reachability properties. It runs a separate set of directory authorities, its own set of Tor routers (called the Blossom network), and uses Tor's flexible path-building to let users view Internet resources from any point in the Blossom network.

Fourth, Tor separates the role of *internal relay* from the role of *exit relay*. That is, some volunteers choose just to relay traffic between Tor users and Tor routers, and others choose to also allow connections to external Internet resources. Because we don't force all volunteers to play both roles, we end up with more relays. This increased diversity in turn is what gives Tor its security: the more options the user has for her first hop, and the more options she has for her last hop, the less likely it is that a given attacker will be watching both ends of her circuit [11]. As a bonus, because our design attracts more internal relays that want to help out but don't want to deal with being an exit relay, we end up providing more options for the first hop—the one most critical to being able to reach the Tor network.

Fifth, Tor is sustainable. Zero-Knowledge Systems offered the commercial but now defunct Freedom Network [2], a design with security comparable to Tor's, but its funding model relied on collecting money from users to pay relay operators. Modern commercial proxy systems similarly need to keep collecting money to support their infrastructure. On the other hand, Tor has built a self-sustaining community of volunteers who donate their time and resources. This community trust is rooted in Tor's open design: we tell the world exactly how Tor works, and we provide all the source code. Users can decide for themselves, or pay any security expert to decide, whether it is safe to use. Further, Tor's modularity as described above, along with its open license, mean that its impact will continue to grow.

Sixth, Tor has an established user base of hundreds of thousands of people from around the world. This diversity of users contributes to sustainability as above: Tor is used by ordinary citizens, activists, corporations, law enforcement, and even government and military users, and they can only achieve their security goals by blending together in the same network [1, 9]. This user base also provides something else: hundreds of thousands of different and often-changing addresses that we can leverage for our blocking-resistance design.

Finally and perhaps most importantly, Tor provides anonymity and prevents any single server from linking users to their communication partners. Despite initial appearances, *distributed-trust anonymity is critical for anti-censorship efforts.* If any single server can expose dissident bloggers or compile a list of users' behavior, the censors can profitably compromise that server's operator, perhaps by applying economic pressure to their employers, breaking into their computer, pressuring their family (if they have relatives in the censored area), or so on. Furthermore, in designs where any relay can expose its users, the censors can spread suspicion that they are running some of the relays and use this belief to chill use of the network.

We discuss and adapt these components further in Section 5. But first we examine the strengths and weaknesses of other blocking-resistance approaches, so we can expand our repertoire of building blocks and ideas.

4 Current proxy solutions

Relay-based blocking-resistance schemes generally have two main components: a relay component and a discovery component. The relay part encompasses the process of establishing a connection, sending traffic back and forth, and so on—everything that's done once the user knows where she's going to connect. Discovery is the step before that: the process of finding one or more usable relays.

For example, we can divide the pieces of Tor in the previous section into the process of building paths and sending traffic over them (relay) and the process of learning from the directory servers about what routers are available (discovery). With this distinction in mind, we now examine several categories of relay-based schemes.

4.1 Centrally-controlled shared proxies

Existing commercial anonymity solutions (like Anonymizer.com) are based on a set of single-hop proxies. In these systems, each user connects to a single proxy, which then relays traffic between the user and her destination. These public proxy systems are typically characterized by two features: they control and operate the proxies centrally, and many different users get assigned to each proxy.

In terms of the relay component, single proxies provide weak security compared to systems that distribute trust over multiple relays, since a compromised proxy can trivially observe all of its users' actions, and an eavesdropper only needs to watch a single proxy to perform timing correlation attacks against all its users' traffic and thus learn where everyone is connecting. Worse, all users need to trust the proxy company to have good security itself as well as to not reveal user activities.

On the other hand, single-hop proxies are easier to deploy, and they can provide better performance than distributed-trust designs like Tor, since traffic only goes through one relay. They're also more convenient from the user's perspective—since users entirely trust the proxy, they can just use their web browser directly.

Whether public proxy schemes are more or less scalable than Tor is still up for debate: commercial anonymity systems can use some of their revenue to provision more bandwidth as they grow, whereas volunteer-based anonymity systems can attract thousands of fast relays to spread the load.

The discovery piece can take several forms. Most commercial anonymous proxies have one or a handful of commonly known websites, and their users log in to those websites and relay their traffic through them. When these websites get blocked (generally soon after the company becomes

popular), if the company cares about users in the blocked areas, they start renting lots of disparate IP addresses and rotating through them as they get blocked. They notify their users of new addresses (by email, for example). It's an arms race, since attackers can sign up to receive the email too, but operators have one nice trick available to them: because they have a list of paying subscribers, they can notify certain subscribers about updates earlier than others.

Access control systems on the proxy let them provide service only to users with certain characteristics, such as paying customers or people from certain IP address ranges.

Discovery in the face of a government-level firewall is a complex and unsolved topic, and we're stuck in this same arms race ourselves; we explore it in more detail in Section 7. But first we examine the other end of the spectrum—getting volunteers to run the proxies, and telling only a few people about each proxy.

4.2 Independent personal proxies

Personal proxies such as Circumventor [18] and CGIProxy [23] use the same technology as the public ones as far as the relay component goes, but they use a different strategy for discovery. Rather than managing a few centralized proxies and constantly getting new addresses for them as the old addresses are blocked, they aim to have a large number of entirely independent proxies, each managing its own (much smaller) set of users.

As the Circumventor site explains, "You don't actually install the Circumventor *on* the computer that is blocked from accessing Web sites. You, or a friend of yours, has to install the Circumventor on some *other* machine which is not censored."

This tactic has great advantages in terms of blocking-resistance—recall our assumption in Section 2 that the attention a system attracts from the attacker is proportional to its number of users and level of publicity. If each proxy only has a few users, and there is no central list of proxies, most of them will never get noticed by the censors.

On the other hand, there's a huge scalability question that so far has prevented these schemes from being widely useful: how does the fellow in China find a person in Ohio who will run a Circumventor for him? In some cases he may know and trust some people on the outside, but in many cases he's just out of luck. Just as hard, how does a new volunteer in Ohio find a person in China who needs it?

This challenge leads to a hybrid design—centrally-distributed personal proxies—which we will investigate in more detail in Section 7.

4.3 Open proxies

Yet another currently used approach to bypassing firewalls is to locate open and misconfigured proxies on the Internet. A quick Google search for "open proxy list" yields a wide variety of freely available lists of HTTP, HTTPS, and SOCKS proxies. Many small companies have sprung up providing more refined lists to paying customers.

There are some downsides to using these open proxies though. First, the proxies are of widely varying quality in terms of bandwidth and stability, and many of them are entirely unreachable. Second, unlike networks of volunteers like Tor, the legality of routing traffic through these proxies is questionable: it's widely believed that most of them don't realize what they're offering, and probably wouldn't allow it if they realized. Third, in many cases the connection to the proxy is unencrypted, so firewalls that filter based on keywords in IP packets will not be hindered. And last, many users are suspicious that some open proxies are a little *too* convenient: are they run by the adversary, in which case they get to monitor all the user's requests just as single-hop proxies can?

A distributed-trust design like Tor resolves each of these issues for the relay component, but a constantly changing set of thousands of open relays is clearly a useful idea for a discovery component.

For example, users might be able to make use of these proxies to bootstrap their first introduction into the Tor network.

4.4 Blocking resistance and JAP

Köpsell and Hilling's Blocking Resistance design [20] is probably the closest related work, and is the starting point for the design in this paper. In this design, the JAP anonymity system [3] is used as a base instead of Tor. Volunteers operate a large number of access points that relay traffic to the core JAP network, which in turn anonymizes users' traffic. The software to run these relays is, as in our design, included in the JAP client software and enabled only when the user decides to enable it. Discovery is handled with a CAPTCHA-based mechanism; users prove that they aren't an automated process, and are given the address of an access point. (The problem of a determined attacker with enough manpower to launch many requests and enumerate all the access points is not considered in depth.) There is also some suggestion that information about access points could spread through existing social networks.

4.5 Infranet

The Infranet design [14] uses one-hop relays to deliver web content, but disguises its communications as ordinary HTTP traffic. Requests are split into multiple requests for URLs on the relay, which then encodes its responses in the content it returns. The relay needs to be an actual website with plausible content and a number of URLs which the user might want to access—if the Infranet software produced its own cover content, it would be far easier for censors to identify. To keep the censors from noticing that cover content changes depending on what data is embedded, Infranet needs the cover content to have an innocuous reason for changing frequently: the paper recommends watermarked images and webcams.

The attacker and relay operators in Infranet's threat model are significantly different than in ours. Unlike our attacker, Infranet's censor can't be bypassed with encrypted traffic (presumably because the censor blocks encrypted traffic, or at least considers it suspicious), and has more computational resources to devote to each connection than ours (so it can notice subtle patterns over time). Unlike our bridge operators, Infranet's operators (and users) have more bandwidth to spare; the overhead in typical steganography schemes is far higher than Tor's.

The Infranet design does not include a discovery element. Discovery, however, is a critical point: if whatever mechanism allows users to learn about relays also allows the censor to do so, he can trivially discover and block their addresses, even if the steganography would prevent mere traffic observation from revealing the relays' addresses.

4.6 RST-evasion and other packet-level tricks

In their analysis of China's firewall's content-based blocking, Clayton, Murdoch and Watson discovered that rather than blocking all packets in a TCP streams once a forbidden word was noticed, the firewall was simply forging RST packets to make the communicating parties believe that the connection was closed [7]. They proposed altering operating systems to ignore forged RST packets.

Other packet-level responses to filtering include splitting sensitive words across multiple TCP packets, so that the censors' firewalls can't notice them without performing expensive stream reconstruction [27]. This technique relies on the same insight as our weak steganography assumption.

4.7 Internal caching networks

Freenet [6] is an anonymous peer-to-peer data store. Analyzing Freenet's security can be difficult, as its design is in flux as new discovery and routing mechanisms are proposed, and no complete

specification has (to our knowledge) been written. Freenet servers relay requests for specific content (indexed by a digest of the content) "toward" the server that hosts it, and then cache the content as it follows the same path back to the requesting user. If Freenet's routing mechanism is successful in allowing nodes to learn about each other and route correctly even as some node-to-node links are blocked by firewalls, then users inside censored areas can ask a local Freenet server for a piece of content, and get an answer without having to connect out of the country at all. Of course, operators of servers inside the censored area can still be targeted, and the addresses of external servers can still be blocked.

4.8 Skype

The popular Skype voice-over-IP software uses multiple techniques to tolerate restrictive networks, some of which allow it to continue operating in the presence of censorship. By switching ports and using encryption, Skype attempts to resist trivial blocking and content filtering. Even if no encryption were used, it would still be expensive to scan all voice traffic for sensitive words. Also, most current keyloggers are unable to store voice traffic. Nevertheless, Skype can still be blocked, especially at its central login server.

4.9 Tor itself

And last, we include Tor itself in the list of current solutions to firewalls. Tens of thousands of people use Tor from countries that routinely filter their Internet. Tor's website has been blocked in most of them. But why hasn't the Tor network been blocked yet?

We have several theories. The first is the most straightforward: tens of thousands of people are simply too few to matter. It may help that Tor is perceived to be for experts only, and thus not worth attention yet. The more subtle variant on this theory is that we've positioned Tor in the public eye as a tool for retaining civil liberties in more free countries, so perhaps blocking authorities don't view it as a threat. (We revisit this idea when we consider whether and how to publicize a Tor variant that improves blocking-resistance—see Section 9.5 for more discussion.)

The broader explanation is that the maintenance of most government-level filters is aimed at stopping widespread information flow and appearing to be in control, not by the impossible goal of blocking all possible ways to bypass censorship. Censors realize that there will always be ways for a few people to get around the firewall, and as long as Tor has not publically threatened their control, they see no urgent need to block it yet.

We should recognize that we're *already* in the arms race. These constraints can give us insight into the priorities and capabilities of our various attackers.

5 The relay component of our blocking-resistant design

Section 3 describes many reasons why Tor is well-suited as a building block in our context, but several changes will allow the design to resist blocking better. The most critical changes are to get more relay addresses, and to distribute them to users differently.

5.1 Bridge relays

Today, Tor servers operate on less than a thousand distinct IP addresses; an adversary could enumerate and block them all with little trouble. To provide a means of ingress to the network, we need a larger set of entry points, most of which an adversary won't be able to enumerate easily. Fortunately, we have such a set: the Tor users.

Hundreds of thousands of people around the world use Tor. We can leverage our already self-selected user base to produce a list of thousands of frequently-changing IP addresses. Specifically, we can give them a little button in the GUI that says "Tor for Freedom", and users who click the button will turn into *bridge relays* (or just *bridges* for short). They can rate limit relayed connections to 10 KB/s (almost nothing for a broadband user in a free country, but plenty for a user who otherwise has no access at all), and since they are just relaying bytes back and forth between blocked users and the main Tor network, they won't need to make any external connections to Internet sites. Because of this separation of roles, and because we're making use of software that the volunteers have already installed for their own use, we expect our scheme to attract and maintain more volunteers than previous schemes.

As usual, there are new anonymity and security implications from running a bridge relay, particularly from letting people relay traffic through your Tor client; but we leave this discussion for Section 8.

5.2 The bridge directory authority

How do the bridge relays advertise their existence to the world? We introduce a second new component of the design: a specialized directory authority that aggregates and tracks bridges. Bridge relays periodically publish server descriptors (summaries of their keys, locations, etc, signed by their long-term identity key), just like the relays in the "main" Tor network, but in this case they publish them only to the bridge directory authorities.

The main difference between bridge authorities and the directory authorities for the main Tor network is that the main authorities provide a list of every known relay, but the bridge authorities only give out a server descriptor if you already know its identity key. That is, you can keep up-to-date on a bridge's location and other information once you know about it, but you can't just grab a list of all the bridges.

The identity key, IP address, and directory port for each bridge authority ship by default with the Tor software, so the bridge relays can be confident they're publishing to the right location, and the blocked users can establish an encrypted authenticated channel. See Section 8.5 for more discussion of the public key infrastructure and trust chain.

Bridges use Tor to publish their descriptors privately and securely, so even an attacker monitoring the bridge directory authority's network can't make a list of all the addresses contacting the authority. Bridges may publish to only a subset of the authorities, to limit the potential impact of an authority compromise.

5.3 Putting them together

If a blocked user knows the identity keys of a set of bridge relays, and he has correct address information for at least one of them, he can use that one to make a secure connection to the bridge authority and update his knowledge about the other bridge relays. He can also use it to make secure connections to the main Tor network and directory servers, so he can build circuits and connect to the rest of the Internet. All of these updates happen in the background: from the blocked user's perspective, he just accesses the Internet via his Tor client like always.

So now we've reduced the problem from how to circumvent the firewall for all transactions (and how to know that the pages you get have not been modified by the local attacker) to how to learn about a working bridge relay.

There's another catch though. We need to make sure that the network traffic we generate by simply connecting to a bridge relay doesn't stand out too much.

6 Hiding Tor's network signatures

Currently, Tor uses two protocols for its network communications. The main protocol uses TLS for encrypted and authenticated communication between Tor instances. The second protocol is standard HTTP, used for fetching directory information. All Tor servers listen on their "ORPort" for TLS connections, and some of them opt to listen on their "DirPort" as well, to serve directory information. Tor servers choose whatever port numbers they like; the server descriptor they publish to the directory tells users where to connect.

One format for communicating address information about a bridge relay is its IP address and DirPort. From there, the user can ask the bridge's directory cache for an up-to-date copy of its server descriptor, and learn its current circuit keys, its ORPort, and so on.

However, connecting directly to the directory cache involves a plaintext HTTP request. A censor could create a network signature for the request and/or its response, thus preventing these connections. To resolve this vulnerability, we've modified the Tor protocol so that users can connect to the directory cache via the main Tor port—they establish a TLS connection with the bridge as normal, and then send a special "begindir" relay command to establish an internal connection to its directory cache.

Therefore a better way to summarize a bridge's address is by its IP address and ORPort, so all communications between the client and the bridge will use ordinary TLS. But there are other details that need more investigation.

What port should bridges pick for their ORPort? We currently recommend that they listen on port 443 (the default HTTPS port) if they want to be most useful, because clients behind standard firewalls will have the best chance to reach them. Is this the best choice in all cases, or should we encourage some fraction of them pick random ports, or other ports commonly permitted through firewalls like 53 (DNS) or 110 (POP)? Or perhaps we should use other ports where TLS traffic is expected, like 993 (IMAPS) or 995 (POP3S). We need more research on our potential users, and their current and anticipated firewall restrictions.

Furthermore, we need to look at the specifics of Tor's TLS handshake. Right now Tor uses some predictable strings in its TLS handshakes. For example, it sets the X.509 organizationName field to "Tor", and it puts the Tor server's nickname in the certificate's commonName field. We should tweak the handshake protocol so it doesn't rely on any unusual details in the certificate, yet it remains secure; the certificate itself should be made to resemble an ordinary HTTPS certificate. We should also try to make our advertised cipher-suites closer to what an ordinary web server would support.

Tor's TLS handshake uses two-certificate chains: one certificate contains the self-signed identity key for the router, and the second contains a current TLS key, signed by the identity key. We use these to authenticate that we're talking to the right router, and to limit the impact of TLS-key exposure. Most (though far from all) consumer-oriented HTTPS services provide only a single certificate. These extra certificates may help identify Tor's TLS handshake; instead, bridges should consider using only a single TLS key certificate signed by their identity key, and providing the full value of the identity key in an early handshake cell. More significantly, Tor currently has all clients present certificates, so that clients are harder to distinguish from servers. But in a blocking-resistance environment, clients should not present certificates at all.

Last, what if the adversary starts observing the network traffic even more closely? Even if our TLS handshake looks innocent, our traffic timing and volume still look different than a user making a secure web connection to his bank. The same techniques used in the growing trend to build tools to recognize encrypted Bittorrent traffic could be used to identify Tor communication and recognize bridge relays. Rather than trying to look like encrypted web traffic, we may be better off trying to blend with some other encrypted network protocol. The first step is to compare typical network behavior for a Tor client to typical network behavior for various other protocols. This statistical cat-and-mouse game is made more complex by the fact that Tor transports a variety of protocols, and we'll want to automatically handle web browsing differently from, say, instant messaging.

6.1 Identity keys as part of addressing information

We have described a way for the blocked user to bootstrap into the network once he knows the IP address and ORPort of a bridge. What about local spoofing attacks? That is, since we never learned an identity key fingerprint for the bridge, a local attacker could intercept our connection and pretend to be the bridge we had in mind. It turns out that giving false information isn't that bad—since the Tor client ships with trusted keys for the bridge directory authority and the Tor network directory authorities, the user can learn whether he's being given a real connection to the bridge authorities or not. (After all, if the adversary intercepts every connection the user makes and gives him a bad connection each time, there's nothing we can do.)

What about anonymity-breaking attacks from observing traffic, if the blocked user doesn't start out knowing the identity key of his intended bridge? The vulnerabilities aren't so bad in this case either—the adversary could do similar attacks just by monitoring the network traffic.

Once the Tor client has fetched the bridge's server descriptor, it should remember the identity key fingerprint for that bridge relay. Thus if the bridge relay moves to a new IP address, the client can query the bridge directory authority to look up a fresh server descriptor using this fingerprint.

So we've shown that it's *possible* to bootstrap into the network just by learning the IP address and ORPort of a bridge, but are there situations where it's more convenient or more secure to learn the bridge's identity fingerprint as well as instead, while bootstrapping? We keep that question in mind as we next investigate bootstrapping and discovery.

7 Discovering working bridge relays

Tor's modular design means that we can develop a better relay component independently of developing the discovery component. This modularity's great promise is that we can pick any discovery approach we like; but the unfortunate fact is that we have no magic bullet for discovery. We're in the same arms race as all the other designs we described in Section 4.

In this section we describe a variety of approaches to adding discovery components for our design.

7.1 Bootstrapping: finding your first bridge.

In Section 5.3, we showed that a user who knows a working bridge address can use it to reach the bridge authority and to stay connected to the Tor network. But how do new users reach the bridge authority in the first place? After all, the bridge authority will be one of the first addresses that a censor blocks.

First, we should recognize that most government firewalls are not perfect. That is, they may allow connections to Google cache or some open proxy servers, or they let file-sharing traffic, Skype, instant messaging, or World-of-Warcraft connections through. Different users will have different mechanisms for bypassing the firewall initially. Second, we should remember that most people don't operate in a vacuum; users will hopefully know other people who are in other situations or have other resources available. In the rest of this section we develop a toolkit of different options and mechanisms, so that we can enable users in a diverse set of contexts to bootstrap into the system.

(For users who can't use any of these techniques, hopefully they know a friend who can—for example, perhaps the friend already knows some bridge relay addresses. If they can't get around it at all, then we can't help them—they should go meet more people or learn more about the technology running the firewall in their area.)

By deploying all the schemes in the toolkit at once, we let bridges and blocked users employ the discovery approach that is most appropriate for their situation.

7.2 Independent bridges, no central discovery

The first design is simply to have no centralized discovery component at all. Volunteers run bridges, and we assume they have some blocked users in mind and communicate their address information to them out-of-band (for example, through Gmail). This design allows for small personal bridges that have only one or a handful of users in mind, but it can also support an entire community of users. For example, Citizen Lab's upcoming Psiphon single-hop proxy tool [13] plans to use this *social network* approach as its discovery component.

There are several ways to do bootstrapping in this design. In the simple case, the operator of the bridge informs each chosen user about his bridge's address information and/or keys. A different approach involves blocked users introducing new blocked users to the bridges they know. That is, somebody in the blocked area can pass along a bridge's address to somebody else they trust. This scheme brings in appealing but complex game theoretic properties: the blocked user making the decision has an incentive only to delegate to trustworthy people, since an adversary who learns the bridge's address and filters it makes it unavailable for both of them. Also, delegating known bridges to members of your social network can be dangerous: an the adversary who can learn who knows which bridges may be able to reconstruct the social network.

Note that a central set of bridge directory authorities can still be compatible with a decentralized discovery process. That is, how users first learn about bridges is entirely up to the bridges, but the process of fetching up-to-date descriptors for them can still proceed as described in Section 5. Of course, creating a central place that knows about all the bridges may not be smart, especially if every other piece of the system is decentralized. Further, if a user only knows about one bridge and he loses track of it, it may be quite a hassle to reach the bridge authority. We address these concerns next.

7.3 Families of bridges, no central discovery

Because the blocked users are running our software too, we have many opportunities to improve usability or robustness. Our second design builds on the first by encouraging volunteers to run several bridges at once (or coordinate with other bridge volunteers), such that some of the bridges are likely to be available at any given time.

The blocked user's Tor client would periodically fetch an updated set of recommended bridges from any of the working bridges. Now the client can learn new additions to the bridge pool, and can expire abandoned bridges or bridges that the adversary has blocked, without the user ever needing to care. To simplify maintenance of the community's bridge pool, each community could run its own bridge directory authority—reachable via the available bridges, and also mirrored at each bridge.

7.4 Public bridges with central discovery

What about people who want to volunteer as bridges but don't know any suitable blocked users? What about people who are blocked but don't know anybody on the outside? Here we describe how to make use of these *public bridges* in a way that still makes it hard for the attacker to learn all of them.

The basic idea is to divide public bridges into a set of pools based on identity key. Each pool corresponds to a *distribution strategy*: an approach to distributing its bridge addresses to users. Each strategy is designed to exercise a different scarce resource or property of the user.

How do we divide bridges between these strategy pools such that they're evenly distributed and the allocation is hard to influence or predict, but also in a way that's amenable to creating more strategies later on without reshuffling all the pools? We assign a given bridge to a strategy pool by hashing the bridge's identity key along with a secret that only the bridge authority knows: the first n bits of this hash dictate the strategy pool number, where n is a parameter that describes how many

strategy pools we want at this point. We choose $n = 3$ to start, so we divide bridges between 8 pools; but as we later invent new distribution strategies, we can increment n to split the 8 into 16. Since a bridge can't predict the next bit in its hash, it can't anticipate which identity key will correspond to a certain new pool when the pools are split. Further, since the bridge authority doesn't provide any feedback to the bridge about which strategy pool it's in, an adversary who signs up bridges with the goal of filling a certain pool [12] will be hindered.

The first distribution strategy (used for the first pool) publishes bridge addresses in a time-release fashion. The bridge authority divides the available bridges into partitions, and each partition is deterministically available only in certain time windows. That is, over the course of a given time slot (say, an hour), each requester is given a random bridge from within that partition. When the next time slot arrives, a new set of bridges from the pool are available for discovery. Thus some bridge address is always available when a new user arrives, but to learn about all bridges the attacker needs to fetch all new addresses at every new time slot. By varying the length of the time slots, we can make it harder for the attacker to guess when to check back. We expect these bridges will be the first to be blocked, but they'll help the system bootstrap until they *do* get blocked. Further, remember that we're dealing with different blocking regimes around the world that will progress at different rates—so this pool will still be useful to some users even as the arms races progress.

The second distribution strategy publishes bridge addresses based on the IP address of the requesting user. Specifically, the bridge authority will divide the available bridges in the pool into a bunch of partitions (as in the first distribution scheme), hash the requester's IP address with a secret of its own (as in the above allocation scheme for creating pools), and give the requester a random bridge from the appropriate partition. To raise the bar, we should discard the last octet of the IP address before inputting it to the hash function, so an attacker who only controls a single "/24" network only counts as one user. A large attacker like China will still be able to control many addresses, but the hassle of establishing connections from each network (or spoofing TCP connections) may still slow them down. Similarly, as a special case, we should treat IP addresses that are Tor exit nodes as all being on the same network.

The third strategy combines the time-based and location-based strategies to further constrain and rate-limit the available bridge addresses. Specifically, the bridge address provided in a given time slot to a given network location is deterministic within the partition, rather than chosen randomly each time from the partition. Thus, repeated requests during that time slot from a given network are given the same bridge address as the first request.

The fourth strategy is based on Circumventor's discovery strategy. The Circumventor project, realizing that its adoption will remain limited if it has no central coordination mechanism, has started a mailing list to distribute new proxy addresses every few days. From experimentation it seems they have concluded that sending updates every three or four days is sufficient to stay ahead of the current attackers.

The fifth strategy provides an alternative approach to a mailing list: users provide an email address and receive an automated response listing an available bridge address. We could limit one response per email address. To further rate limit queries, we could require a CAPTCHA solution in each case too. In fact, we wouldn't need to implement the CAPTCHA on our side: if we only deliver bridge addresses to Yahoo or GMail addresses, we can leverage the rate-limiting schemes that other parties already impose for account creation.

The sixth strategy ties in the social network design with public bridges and a reputation system. We pick some seeds—trusted people in blocked areas—and give them each a few dozen bridge addresses and a few *delegation tokens*. We run a website next to the bridge authority, where users can log in (they connect via Tor, and they don't need to provide actual identities, just persistent pseudonyms). Users can delegate trust to other people they know by giving them a token, which can be exchanged for a new account on the website. Accounts in "good standing" then accrue new bridge addresses and new tokens. As usual, reputation schemes bring in a host of new complexities [10]: how

do we decide that an account is in good standing? We could tie reputation to whether the bridges they're told about have been blocked—see Section 7.7 below for initial thoughts on how to discover whether bridges have been blocked. We could track reputation between accounts (if you delegate to somebody who screws up, it impacts you too), or we could use blinded delegation tokens [5] to prevent the website from mapping the seeds' social network. We put off deeper discussion of the social network reputation strategy for future work.

Pools seven and eight are held in reserve, in case our currently deployed tricks all fail at once and the adversary blocks all those bridges—so we can adapt and move to new approaches quickly, and have some bridges immediately available for the new schemes. New strategies might be based on some other scarce resource, such as relaying traffic for others or other proof of energy spent. (We might also worry about the incentives for bridges that sign up and get allocated to the reserve pools: will they be unhappy that they're not being used? But this is a transient problem: if Tor users are bridges by default, nobody will mind not being used yet. See also Section 9.4.)

7.5 Public bridges with coordinated discovery

We presented the above discovery strategies in the context of a single bridge directory authority, but in practice we will want to distribute the operations over several bridge authorities—a single point of failure or attack is a bad move. The first answer is to run several independent bridge directory authorities, and bridges gravitate to one based on their identity key. The better answer would be some federation of bridge authorities that work together to provide redundancy but don't introduce new security issues. We could even imagine designs where the bridge authorities have encrypted versions of the bridge's server descriptors, and the users learn a decryption key that they keep private when they first hear about the bridge—this way the bridge authorities would not be able to learn the IP address of the bridges.

We leave this design question for future work.

7.6 Assessing whether bridges are useful

Learning whether a bridge is useful is important in the bridge authority's decision to include it in responses to blocked users. For example, if we end up with a list of thousands of bridges and only a few dozen of them are reachable right now, most blocked users will not end up knowing about working bridges.

There are three components for assessing how useful a bridge is. First, is it reachable from the public Internet? Second, what proportion of the time is it available? Third, is it blocked in certain jurisdictions?

The first component can be tested just as we test reachability of ordinary Tor servers. Specifically, the bridges do a self-test—connect to themselves via the Tor network—before they are willing to publish their descriptor, to make sure they're not obviously broken or misconfigured. Once the bridges publish, the bridge authority also tests reachability to make sure they're not confused or outright lying.

The second component can be measured and tracked by the bridge authority. By doing periodic reachability tests, we can get a sense of how often the bridge is available. More complex tests will involve bandwidth-intensive checks to force the bridge to commit resources in order to be counted as available. We need to evaluate how the relationship of uptime percentage should weigh into our choice of which bridges to advertise. We leave this to future work.

The third component is perhaps the trickiest: with many different adversaries out there, how do we keep track of which adversaries have blocked which bridges, and how do we learn about new blocks as they occur? We examine this problem next.

7.7 How do we know if a bridge relay has been blocked?

There are two main mechanisms for testing whether bridges are reachable from inside each blocked area: active testing via users, and passive testing via bridges.

In the case of active testing, certain users inside each area sign up as testing relays. The bridge authorities can then use a Blossom-like [16] system to build circuits through them to each bridge and see if it can establish the connection. But how do we pick the users? If we ask random users to do the testing (or if we solicit volunteers from the users), the adversary should sign up so he can enumerate the bridges we test. Indeed, even if we hand-select our testers, the adversary might still discover their location and monitor their network activity to learn bridge addresses.

Another answer is not to measure directly, but rather let the bridges report whether they're being used. Specifically, bridges should install a GeoIP database such as the public IP-To-Country list [19], and then periodically report to the bridge authorities which countries they're seeing use from. This data would help us track which countries are making use of the bridge design, and can also let us learn about new steps the adversary has taken in the arms race. (The compressed GeoIP database is only several hundred kilobytes, and we could even automate the update process by serving it from the bridge authorities.) More analysis of this passive reachability testing design is needed to resolve its many edge cases: for example, if a bridge stops seeing use from a certain area, does that mean the bridge is blocked or does that mean those users are asleep?

There are many more problems with the general concept of detecting whether bridges are blocked. First, different zones of the Internet are blocked in different ways, and the actual firewall jurisdictions do not match country borders. Our bridge scheme could help us map out the topology of the censored Internet, but this is a huge task. More generally, if a bridge relay isn't reachable, is that because of a network block somewhere, because of a problem at the bridge relay, or just a temporary outage somewhere in between? And last, an attacker could poison our bridge database by signing up already-blocked bridges. In this case, if we're stingy giving out bridge addresses, users in that country won't learn working bridges.

All of these issues are made more complex when we try to integrate this testing into our social network reputation system above. Since in that case we punish or reward users based on whether bridges get blocked, the adversary has new attacks to trick or bog down the reputation tracking. Indeed, the bridge authority doesn't even know what zone the blocked user is in, so do we blame him for any possible censored zone, or what?

Clearly more analysis is required. The eventual solution will probably involve a combination of passive measurement via GeoIP and active measurement from trusted testers. More generally, we can use the passive feedback mechanism to track usage of the bridge network as a whole—which would let us respond to attacks and adapt the design, and it would also let the general public track the progress of the project.

7.8 Advantages of deploying all solutions at once

For once, we're not in the position of the defender: we don't have to defend against every possible filtering scheme; we just have to defend against at least one. On the flip side, the attacker is forced to guess how to allocate his resources to defend against each of these discovery strategies. So by deploying all of our strategies at once, we not only increase our chances of finding one that the adversary has difficulty blocking, but we actually make *all* of the strategies more robust in the face of an adversary with limited resources.

8 Security considerations

8.1 Possession of Tor in oppressed areas

Many people speculate that installing and using a Tor client in areas with particularly extreme firewalls is a high risk—and the risk increases as the firewall gets more restrictive. This notion certain has merit, but there's a counter pressure as well: as the firewall gets more restrictive, more ordinary people behind it end up using Tor for more mainstream activities, such as learning about Wall Street prices or looking at pictures of women's ankles. So as the restrictive firewall pushes up the number of Tor users, the "typical" Tor user becomes more mainstream, and therefore mere use or possession of the Tor software is not so surprising.

It's hard to say which of these pressures will ultimately win out, but we should keep both sides of the issue in mind.

8.2 Observers can tell who is publishing and who is reading

Tor encrypts traffic on the local network, and it obscures the eventual destination of the communication, but it doesn't do much to obscure the traffic volume. In particular, a user publishing a home video will have a different network signature than a user reading an online news article. Based on our assumption in Section 2 that users who publish material are in more danger, should we work to improve Tor's security in this situation?

In the general case this is an extremely challenging task: effective *end-to-end traffic confirmation attacks* are known where the adversary observes the origin and the destination of traffic and confirms that they are part of the same communication [8, 24]. Related are *website fingerprinting attacks*, where the adversary downloads a few hundred popular websites, makes a set of "signatures" for each site, and then observes the target Tor client's traffic to look for a match [4, 21]. But can we do better against a limited adversary who just does coarse-grained sweeps looking for unusually prolific publishers?

One answer is for bridge users to automatically send bursts of padding traffic periodically. (This traffic can be implemented in terms of long-range drop cells, which are already part of the Tor specification.) Of course, convincingly simulating an actual human publishing interesting content is a difficult arms race, but it may be worthwhile to at least start the race. More research remains.

8.3 Anonymity effects from acting as a bridge relay

Against some attacks, relaying traffic for others can improve anonymity. The simplest example is an attacker who owns a small number of Tor servers. He will see a connection from the bridge, but he won't be able to know whether the connection originated there or was relayed from somebody else. More generally, the mere uncertainty of whether the traffic originated from that user may be helpful.

There are some cases where it doesn't seem to help: if an attacker can watch all of the bridge's incoming and outgoing traffic, then it's easy to learn which connections were relayed and which started there. (In this case he still doesn't know the final destinations unless he is watching them too, but in this case bridges are no better off than if they were an ordinary client.)

There are also some potential downsides to running a bridge. First, while we try to make it hard to enumerate all bridges, it's still possible to learn about some of them, and for some people just the fact that they're running one might signal to an attacker that they place a higher value on their anonymity. Second, there are some more esoteric attacks on Tor relays that are not as well-understood or well-tested—for example, an attacker may be able to "observe" whether the bridge is sending traffic even if he can't actually watch its network, by relaying traffic through it and noticing changes in traffic timing [25]. On the other hand, it may be that limiting the bandwidth the bridge

is willing to relay will allow this sort of attacker to determine if it's being used as a bridge but not easily learn whether it is adding traffic of its own.

We also need to examine how entry guards fit in. Entry guards (a small set of nodes that are always used for the first step in a circuit) help protect against certain attacks where the attacker runs a few Tor servers and waits for the user to choose these servers as the beginning and end of her circuit[2]. If the blocked user doesn't use the bridge's entry guards, then the bridge doesn't gain as much cover benefit. On the other hand, what design changes are needed for the blocked user to use the bridge's entry guards without learning what they are (this seems hard), and even if we solve that, do they then need to use the guards' guards and so on down the line?

It is an open research question whether the benefits of running a bridge outweigh the risks. A lot of the decision rests on which attacks the users are most worried about. For most users, we don't think running a bridge relay will be that damaging, and it could help quite a bit.

8.4 Trusting local hardware: Internet cafes and LiveCDs

Assuming that users have their own trusted hardware is not always reasonable.

For Internet cafe Windows computers that let you attach your own USB key, a USB-based Tor image would be smart. There's Torpark, and hopefully there will be more thoroughly analyzed and trustworthy options down the road. Worries remain about hardware or software keyloggers and other spyware, as well as and physical surveillance.

If the system lets you boot from a CD or from a USB key, you can gain a bit more security by bringing a privacy LiveCD with you. (This approach isn't foolproof either of course, since hardware keyloggers and physical surveillance are still a worry).

In fact, LiveCDs are also useful if it's your own hardware, since it's easier to avoid leaving private data and logs scattered around the system.

8.5 The trust chain

Tor's "public key infrastructure" provides a chain of trust to let users verify that they're actually talking to the right servers. There are four pieces to this trust chain.

First, when Tor clients are establishing circuits, at each step they demand that the next Tor server in the path prove knowledge of its private key [11]. This step prevents the first node in the path from just spoofing the rest of the path. Second, the Tor directory authorities provide a signed list of servers along with their public keys—so unless the adversary can control a threshold of directory authorities, he can't trick the Tor client into using other Tor servers. Third, the location and keys of the directory authorities, in turn, is hard-coded in the Tor source code—so as long as the user got a genuine version of Tor, he can know that he is using the genuine Tor network. And last, the source code and other packages are signed with the GPG keys of the Tor developers, so users can confirm that they did in fact download a genuine version of Tor.

In the case of blocked users contacting bridges and bridge directory authorities, the same logic applies in parallel: the blocked users fetch information from both the bridge authorities and the directory authorities for the 'main' Tor network, and they combine this information locally.

How can a user in an oppressed country know that he has the correct key fingerprints for the developers? As with other security systems, it ultimately comes down to human interaction. The keys are signed by dozens of people around the world, and we have to hope that our users have met enough people in the PGP web of trust that they can learn the correct keys. For users that aren't connected to the global security community, though, this question remains a critical weakness.

[2] `http://wiki.noreply.org/noreply/TheOnionRouter/TorFAQ\#EntryGuards`

9 Maintaining reachability

9.1 How many bridge relays should you know about?

The strategies described in Section 7 talked about learning one bridge address at a time. But if most bridges are ordinary Tor users on cable modem or DSL connection, many of them will disappear and/or move periodically. How many bridge relays should a blocked user know about so that she is likely to have at least one reachable at any given point? This is already a challenging problem if we only consider natural churn: the best approach is to see what bridges we attract in reality and measure their churn. We may also need to factor in a parameter for how quickly bridges get discovered and blocked by the attacker; we leave this for future work after we have more deployment experience.

A related question is: if the bridge relays change IP addresses periodically, how often does the blocked user need to fetch updates in order to keep from being cut out of the loop?

Once we have more experience and intuition, we should explore technical solutions to this problem too. For example, if the discovery strategies give out k bridge addresses rather than a single bridge address, perhaps we can improve robustness from the user perspective without significantly aiding the adversary. Rather than giving out a new random subset of k addresses at each point, we could bind them together into *bridge families*, so all users that learn about one member of the bridge family are told about the rest as well.

This scheme may also help defend against attacks to map the set of bridges. That is, if all blocked users learn a random subset of bridges, the attacker should learn about a few bridges, monitor the country-level firewall for connections to them, then watch those users to see what other bridges they use, and repeat. By segmenting the bridge address space, we can limit the exposure of other users.

9.2 Cablemodem users don't usually provide important websites

Another attacker we might be concerned about is that the attacker could just block all DSL and cablemodem network addresses, on the theory that they don't run any important services anyway. If most of our bridges are on these networks, this attack could really hurt.

The first answer is to aim to get volunteers both from traditionally "consumer" networks and also from traditionally "producer" networks. Since bridges don't need to be Tor exit nodes, as we improve our usability it seems quite feasible to get a lot of websites helping out.

The second answer (not as practical) would be to encourage more use of consumer networks for popular and useful Internet services.

A related attack we might worry about is based on large countries putting economic pressure on companies that want to expand their business. For example, what happens if Verizon wants to sell services in China, and China pressures Verizon to discourage its users in the free world from running bridges?

9.3 Scanning resistance: making bridges more subtle

If it's trivial to verify that a given address is operating as a bridge, and most bridges run on a predictable port, then it's conceivable our attacker could scan the whole Internet looking for bridges. (In fact, he can just concentrate on scanning likely networks like cablemodem and DSL services—see Section 9.2 above for related attacks.) It would be nice to slow down this attack. It would be even nicer to make it hard to learn whether we're a bridge without first knowing some secret. We call this general property *scanning resistance*, and it goes along with normalizing Tor's TLS handshake and network signature.

We could provide a password to the blocked user, and she (or her Tor client) provides a nonced hash of this password when she connects. We'd need to give her an ID key for the bridge too (in

addition to the IP address and port—see Section 6.1), and wait to present the password until we've finished the TLS handshake, else it would look unusual. If Alice can authenticate the bridge before she tries to send her password, we can resist an adversary who pretends to be the bridge and launches a man-in-the-middle attack to learn the password. But even if she can't, we resist against widespread scanning.

Another approach would be some kind of ID-based knocking protocol, where the bridge only answers after some predefined set of connections or interactions. The bridge could even act like an unconfigured HTTPS server ("welcome to the default Apache page") if accessed without the correct authorization.

We might assume that the attacker can recognize HTTPS connections that use self-signed certificates. (This process would be resource-intensive but not out of the realm of possibility.) But even in this case, many popular websites around the Internet use self-signed or just plain broken SSL certificates.

9.4 How to motivate people to run bridge relays

One of the traditional ways to get people to run software that benefits others is to give them motivation to install it themselves. An often suggested approach is to install it as a stunning screensaver so everybody will be pleased to run it. We take a similar approach here, by leveraging the fact that these users are already interested in protecting their own Internet traffic, so they will install and run the software.

Eventually, we may be able to make all Tor users become bridges if they pass their self-reachability tests—the software and installers need more work on usability first, but we're making progress.

In the mean time, we can make a snazzy network graph with Vidalia[3] that emphasizes the connections the bridge user is currently relaying.

9.5 Publicity attracts attention

Many people working on this field want to publicize the existence and extent of censorship concurrently with the deployment of their circumvention software. The easy reason for this two-pronged push is to attract volunteers for running proxies in their systems; but in many cases their main goal is not to focus on actually allowing individuals to circumvent the firewall, but rather to educate the world about the censorship. The media also tries to do its part by broadcasting the existence of each new circumvention system.

But at the same time, this publicity attracts the attention of the censors. We can slow down the arms race by not attracting as much attention, and just spreading by word of mouth. If our goal is to establish a solid social network of bridges and bridge users before the adversary gets involved, does this extra attention work to our disadvantage?

9.6 The Tor website: how to get the software

One of the first censoring attacks against a system like ours is to block the website and make the software itself hard to find. Our system should work well once the user is running an authentic copy of Tor and has found a working bridge, but to get to that point we rely on their individual skills and ingenuity.

Right now, most countries that block access to Tor block only the main website and leave mirrors and the network itself untouched. Falling back on word-of-mouth is always a good last resort, but we should also take steps to make sure it's relatively easy for users to get a copy, such as publicizing the mirrors more and making copies available through other media. See Section 7.1 for more discussion.

[3] `http://vidalia-project.net/`

10 Future designs

10.1 Bridges inside the blocked network too

Assuming actually crossing the firewall is the risky part of the operation, can we have some bridge relays inside the blocked area too, and more established users can use them as relays so they don't need to communicate over the firewall directly at all? A simple example here is to make new blocked users into internal bridges also—so they sign up on the bridge authority as part of doing their query, and we give out their addresses rather than (or along with) the external bridge addresses. This design is a lot trickier because it brings in the complexity of whether the internal bridges will remain available, can maintain reachability with the outside world, etc.

More complex future designs involve operating a separate Tor network inside the blocked area, and using *hidden service bridges*—bridges that can be accessed by users of the internal Tor network but whose addresses are not published or findable, even by these users—to get from inside the firewall to the rest of the Internet. But this design requires directory authorities to run inside the blocked area too, and they would be a fine target to take down the network.

11 Next Steps

Technical solutions won't solve the whole censorship problem. After all, the firewalls in places like China and Iran are *socially* very successful, even if technologies and tricks exist to get around them. However, having a strong technical solution is still necessary as one important piece of the puzzle.

In this paper, we have shown that Tor provides a great set of building blocks to start from. The next steps are to deploy prototype bridges and bridge authorities, implement some of the proposed discovery strategies, and then observe the system in operation and get more intuition about the actual requirements and adversaries we're up against.

References

1. Alessandro Acquisti, Roger Dingledine, and Paul Syverson. On the economics of anonymity. In Rebecca N. Wright, editor, *Financial Cryptography*. Springer-Verlag, LNCS 2742, 2003.
2. Adam Back, Ian Goldberg, and Adam Shostack. Freedom systems 2.1 security issues and analysis. White paper, Zero Knowledge Systems, Inc., May 2001.
3. Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, 2000.
4. George Dean Bissias, Marc Liberatore, and Brian Neil Levine. Privacy vulnerabilities in encrypted http streams. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2005)*, May 2005. `http://prisms.cs.umass.edu/brian/pubs/bissias.liberatore.pet.2005.pdf`.
5. David Chaum. Blind signatures for untraceable payments. In D. Chaum, R.L. Rivest, and A.T. Sherman, editors, *Advances in Cryptology:Proceedings of Crypto 82*, pages 199–203. Plenum Press, 1983.
6. Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability*, pages 46–66. Springer-Verlag, LNCS 2009, July 2000.
7. Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson. Ignoring the great firewall of china. In *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006)*, Cambridge, UK, June 2006. Springer. `http://www.cl.cam.ac.uk/~rnc1/ignoring.pdf`.
8. George Danezis. The traffic analysis of continuous-time mixes. In David Martin and Andrei Serjantov, editors, *Privacy Enhancing Technologies (PET 2004)*, LNCS, May 2004. `http://www.cl.cam.ac.uk/users/gd216/cmm2.pdf`.

9. Roger Dingledine and Nick Mathewson. Anonymity loves company: Usability and the network effect. In *Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006)*, Cambridge, UK, June 2006. http://freehaven.net/doc/wupss04/usability.pdf.
10. Roger Dingledine, Nick Mathewson, and Paul Syverson. Reputation in P2P Anonymity Systems. In *Proceedings of Workshop on Economics of Peer-to-Peer Systems*, June 2003. http://freehaven.net/doc/econp2p03/econp2p03.pdf.
11. Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004. http://tor.eff.org/tor-design.pdf.
12. Roger Dingledine and Paul Syverson. Reliable MIX Cascade Networks through Reputation. In Matt Blaze, editor, *Financial Cryptography*. Springer-Verlag, LNCS 2357, 2002.
13. Ronald Deibert et al. Psiphon. http://psiphon.civisec.org/.
14. Nick Feamster, Magdalena Balazinska, Greg Harfst, Hari Balakrishnan, and David Karger. Infranet: Circumventing web censorship and surveillance. In *Proceedings of the 11th USENIX Security Symposium*, August 2002. http://nms.lcs.mit.edu/~feamster/papers/usenixsec2002.pdf.
15. Gina Fisk, Mike Fisk, Christos Papadopoulos, and Joshua Neil. Eliminating steganography in internet traffic with active wardens. In Fabien Petitcolas, editor, *Information Hiding Workshop (IH 2002)*. Springer-Verlag, LNCS 2578, October 2002.
16. Geoffrey Goodell. *Perspective Access Networks*. PhD thesis, Harvard University, July 2006. http://afs.eecs.harvard.edu/~goodell/thesis.pdf.
17. Geoffrey Goodell and Paul Syverson. The right place at the right time: The use of network location in authentication and abuse prevention, 2006. Submitted.
18. Bennett Haselton. How to install the Circumventor program. http://www.peacefire.org/circumventor/simple-circumventor-instructions.%html.
19. Ip-to-country database. http://ip-to-country.webhosting.info/.
20. Stefan Köpsell and Ulf Hilling. How to achieve blocking resistance for existing systems enabling anonymous web surfing. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2004)*, Washington, DC, USA, October 2004. http://freehaven.net/anonbib/papers/p103-koepsell.pdf.
21. Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew Wright. Timing analysis in low-latency mix-based systems. In Ari Juels, editor, *Financial Cryptography*. Springer-Verlag, LNCS (forthcoming), 2004.
22. Rebecca MacKinnon. Private communication, 2006.
23. James Marshall. CGIProxy: HTTP/FTP Proxy in a CGI Script. http://www.jmarshall.com/tools/cgiproxy/.
24. Nick Mathewson and Roger Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In David Martin and Andrei Serjantov, editors, *Privacy Enhancing Technologies (PET 2004)*, LNCS, May 2004. http://freehaven.net/doc/e2e-traffic/e2e-traffic.pdf.
25. Steven J. Murdoch and George Danezis. Low-cost traffic analysis of tor. In *IEEE Symposium on Security and Privacy*. IEEE CS, May 2005.
26. Steven J. Murdoch and Stephen Lewis. Embedding covert channels into TCP/IP. In Mauro Barni, Jordi Herrera-Joancomartí, Stefan Katzenbeisser, and Fernando Pérez-González, editors, *Information Hiding: 7th International Workshop*, volume 3727 of *LNCS*, pages 247–261, Barcelona, Catalonia (Spain), June 2005. Springer-Verlag.
27. Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., Suite 330, 1201 5th Street S.W, Calgary, Alberta, Canada, T2R-0Y6, 1998.
28. Ethan Zuckerman. We've got to adjust some of our threat models. http://www.ethanzuckerman.com/blog/?p=1019.