Addressing Denial of Service Attacks on Free and Open Communication on the Internet

— Final Report —

Cecylia Bocovich, Roger Dingledine, Alexander Færøy,
Kat Hanna, Philipp Winter, Taylor Yu
kat@torproject.org

Tor Tech Report 2019-05-001
May 31, 2019

Contents

1 Introduction

Around the world, people are trying to safely communicate about their situations and the situations of others. This free and open communication on the internet is critical to the growth of strong communities and societies. At the same time, nation-state actors and others attempt to *deny* this communication, threatening this growth. In this project we extend our past work on the Tor anonymity system and the pluggable transport ecosystem to explore and analyze techniques to detect, understand, and mitigate these denials of service. In our first report [5] for this project, we presented the then-current state of several areas related to combatting internet censorship. In this report, we lay out our roadmaps for research and practical development in some of these areas. While much of this work is planned or in progress, the parts that are completed are marked as such.

This report is organized as follows: Section 2 discusses work needed to improve the BridgeDB service; Section 3 discusses work needed to improve the GetTor service; Section 4 describes what we plan to do to maintain existing, viable pluggable transports and the general pluggable transport framework; Section 5 talks about the Snowflake pluggable transport; Section 6 describes several next generation pluggable transports and our plans for developing and testing them; Section 7 describes steps we can take to improve the user experience for people on poor networks; Section 8 discusses areas to explore to better understand censorship of the Tor network; Section 9.1 outlines how we can help people make better use of Tor bridges;

Section 10 describes extending our community outreach to better serve users in censored locations; Section 11 outlines some of the anti-censorship work planned by the Tails project, which is affiliated with the Tor Project; and Section 12 concludes.

2 Improving the BridgeDB Service

Some censors attempt to block access to the Tor network by blocking the Tor relays listed in the public directory of relays. Tor bridges are relays that are not listed in the public directory. This makes bridges harder to block than regular relays, but creates a challenge: how can we distribute bridge addresses to users while keeping them from censors? There are currently a variety of ways for users to discover bridges: some bridges come pre-configured in Tor Browser; users can request bridges using an ordinary web browser or by email; in Tor Browser 8.0 and later, the moat feature allows a user to request a bridge from within the browser settings and have it configured automatically. In all of these cases, the bridge addresses come from BridgeDB, a database that stores addresses of all of the currently known public bridges.

Several areas of work to improve BridgeDB are detailed below.

2.1 Improve BridegDB's user experience

2.1.1 Localization

We need to localize all aspects of the BridgeDB user flow. In some cases, this will involve investigating why localization that has already been implemented is not working. An example of this is that response emails that should be offered in non-English languages are not. In addition, we need a language switcher on the https://bridges.torproject.org so that users can override geolocation.

2.1.2 Improve CAPTCHA usability

Users requesting bridge addresses from the BridgeDB web interface are required to solve a CAPTCHA before receiving bridge addresses. Our current CAPTCHA system often provides images that contain letters that are very difficult for humans to discern. In addition, we do not currently provide audio CAPTCHAs that would enable users with visual impairments to request bridge addresses from the web interface. We need to address these issues to make BridgeDB's web interface usable for more people.

2.2 Investigate design improvements

2.2.1 Improve the current BridgeDB implementation

We need to consider ways to improve the design of the BridgeDB service. First we need to assess current usage patterns such as which distribution mechanisms are being used, how much each is being used, and where requests are coming from. As part of this work, we should specifically consider whether Yahoo's account creation protections have been weakened and thus whether we should continue to accept requests from that domain.

We should also consider additional domains to accept email requests from, taking into account which censored locations need non-default bridges to connect to the Tor network.

Salmon [6] offers a proxy distribution method that employs an algorithm to automatically identify and prevent malicious users from learning bridge information. We should evaluate whether we could use Salmon to improve how BridgeDB distributes bridges.

2.2.2 Add automated monitoring

The BridgeDB service needs automated monitoring to ensure availability of all aspects of the service. This should include whether the email request mechanism is working, whether it gets updated, and whether the bridge authority is reachable. This falls under the umbrella of adding monitoring for all of our anti-censorship infrastructure.

2.2.3 Monitor the bridge authority COMPLETED

The bridge authority is currently monitored by sysmon.

2.3 Assess and improve the email distribution mechanism

We need to assess whether BridgeDB's email request mechanism is working correctly, and consider ways to improve it. One area to address is that an email reply is not sent if the help text is quoted in the body of the email request.

We should consider partitioning the set of bridges so that BridgeDB distributes distinct subsets in response to requests from each email domain. This would prevent a subverted provider from learning the complete list of bridge addresses. However, this enhancement might also allow an observer to learn a user's email provider by observing which bridge they connect to. We need to consider how to accomplish the goal of protecting bridges while also protecting user privacy.

2.4 Current implementation: status and cleanup COMPLETED

There are a few areas where we cleaned up BridgeDB's configuration and start-up system:

- We added a README file to document BridgeDB administration, including how the repository is organized, as well as how the current instance is configured. This makes it easy to set up another instance if necessary.

- We cleanup up bridgebd.crontab, removing unused lines, cleaning up comments, and otherwise better documenting the configuration.

- We cleaned up bridgedb.conf, removing outdated configuration entries.

- We added a systemd initialization script to facilitate starting and reloading bridgedb.

2.4.1 Address continuous integration error COMPLETED

We use the Travis continuous integration system for BridgeDB. We closed a ticket concerning a temporary error that is no longer happening.

2.5 Improve BridgeDB deployment

2.5.1 Create a software release for BridgeDB

Historically, BridgeDB has been deployed and run by the BridgeDB developers. It would be better to decouple the development from the deployment and administration, so we want to create a software release for BridgeDB. In addition, we could make the release available so that NGOs, for example, could deploy their own instance of BridgeDB to distribute private bridges to their members and allies inside a censored country. For more information on making it easier for NGOs to use private bridges, see Section 9.1.

2.5.2 Document BridgeDB Infrastructure COMPLETED

We documented the configuration of the currently running BridgeDB so that we can replicate it when needed.

2.6 Consider alternative methods of bridge distribution

The traditional way of distributing bridge information is for BridgeDB to give out bridge connection information that is then added to the Tor Browser configuration either manually or automatically.

Snowflake and meek handle bridge information differently at the client side, and we might like to consider a broader application of the Snowflake broker style of distributing bridges. To do so, we need to assess what changes would be required at the client side to make this work.

2.7 Distribute IPv6 bridges

BridgeDB does not currently distribute IPv6 bridge addresses, but IPv6 reachability testing is enabled from the bridge authority and Tor Metrics collects information about IPv6 bridges. We need to investigate why BridgeDB does not distribute these bridges and fix the issue.

2.8 Protect bridges with robust pluggable transports COMPLETED

Many Tor bridges can be used either as regular bridges or as bridges with pluggable transports. In these cases, the pluggable transport is reachable on the same IP address as the regular bridge, but on a different port. As we discussed in Interim Report 1 [5], obfs4 is not susceptible to the active probing attacks that regular bridges and older pluggable transports are. This makes bridges running an obfs4 pluggable transport valuable in certain censored locations that routinely use active probing, such as China.

Recent research [9, 23, 7] has indicated that China has moved from blocking bridges by IP address and port to blocking all connections to a bridge's IP address. This means that when a user in China requests a regular bridge or a bridge with a pluggable transport that is not resistant to active probing, when they try to use it, the censor will learn that it is a bridge and block its IP address. If that bridge also offers the obfs4 pluggable transport on a different port, it will be blocked as well. This is an unfortunate result because of the value of obfs4 bridges for evading censorship in China.

To protect bridges running obfs4 from being discovered and blocked in this way, if a bridge offers obfs4, BridgeDB only distributes its connection information (that is, the IP address and port) when an obfs4 bridge is requested.

2.9 Collect and export statistics

In order to understand how BridgeDB is being used and how we might improve the service, we need to collect statistics. Useful metrics might include the number of requests, the numbers of requests from each country, and the frequency with which each distributor is being used. A first step in completing this work is to evaluate what statistics we are already collecting and how these are meeting our needs.

Specifically, we would like to know how many bridges exist for each pluggable transport. We might be able to collect this information from within BridgeDB or we may prefer to have BridgeDB expose the information in a format that CollectTor [3] can use to integrate the information into our Tor Metrics website.

There are several considerations to address in terms of how we should count the bridges:

- Should we only count bridges that have the *running* flag? Only ones reachable by OONIProbe?
- How should we sanitize the numbers? Round them? Provide a range?
- Do we care about the number of bridges that offer a particular transport or the number of instances?

3 Improving the GetTor Service

In cases where censors block access to the Tor Project website, users need another method for downloading Tor Browser. GetTor, a service that provides a way for a user to get Tor Browser from another site, was developed by an intern participating in Google's Summer of Code. GetTor has gone unmaintained for the past few years, but we now have plans and the necessary resources to improve it and make it a standard Tor service.

GetTor can use multiple channels to receive requests and distribute links. Email and Twitter have been the channels used in the past, but we are exploring new channels, as well. GetTor also uses multiple providers to host the Tor Browser software bundles that users download.

3.1 Assess and improve distribution channels and software providers

3.1.1 Consider more channels and providers

GetTor currently delivers links to the Tor Browser software through email and over Twitter. We need to evaluate using other messaging platforms to deliver links, such as XMPP, Telegram, Tox, Bitmessage, and Facebook.

We currently use Google Drive, Github, and Dropbox to host the Tor Browser software bundles and will be adding Gitlab shortly. We should consider additional providers to host the bundles.

3.1.2 Twitter

There are a few issues related to the Twitter distribution method that need addressing.

- The @get_tor twitter bot is not responding to requests because the Twitter API has changed. We need to update the bot to accommodate these changes.
- We need to get the @get_tor twitter bot verified so that users making requests can be confident that they are making requests through an official Tor Project channel.

3.2 Improve software bundle distribution

3.2.1 Automate updates to providers

In order to ensure that we are always serving links to the latest Tor Browser packages, we need to automate the delivery of the packages to each of the providers. We currently use Google Drive, Github, and Dropbox to host the software bundles and will be adding Gitlab shortly.

As part of this work, we need to include checks to see if a bundle is already stored at a given provider, and whether it should be replaced with a more recent version.

3.2.2 Create more secure cloud accounts

The Github repository is part of the official Tor Project Github account, but the accounts at other providers are currently personal accounts. We need to create official Tor Project accounts for Google Drive and Dropbox and tighten the security of these by adding two-factor authentication to them. This will make it harder for an adversary to hijack the accounts in order to serve inauthentic versions of Tor Browser.

3.2.3 Consider providing links to the alpha bundle

We don't currently make the alpha version of Tor Browser available through GetTor, but we should consider doing so because some users in censored countries might want it. Along with the most recent version, we currently offer the immediately previous version to handle cases where users have older links. Space limitation in our cloud provider accounts may mean this decision is a trade-off between offering this previous version or the alpha version.

3.3 Improve software integrity checks

Because GetTor helps users retrieve software from a variety of locations with varying levels of security, we need to make sure there are easy ways for users to verify the integrity of the software they get.

3.3.1 Improve instructions for checking software bundle integrity

We need to provide simple, straightforward instructions to help users verify the integrity of the software bundles they download. We want to do this by providing a minimal explanation in the body of the message that emphasizes the importance of completing the integrity check, as well as a link to a guide that walks the users through doing the check.

3.3.2 Deliver checksums for the software bundles

Sometimes GetTor needs to deliver the Tor Browser software over insecure channels, for example, if the software bundle is too large to be allowed over gmail. In cases like this, we need to deliver checksums for the software bundles to the user so that they can check the integrity of their bundle. We could do this by making it possible for users to request checksums from GetTor or by automatically sending the appropriate checksum file along with the link to the bundle.

3.3.3 Check integrity of hosted software periodically

We should create an automated script that periodically downloads our software from the cloud providers we use to serve it and checks its integrity. This would alert us in case files are altered or otherwise become corrupted.

3.4 Improve translation coverage

We need to add the GetTor service to our translation process and get it translated into our standard languages. It is especially important that we have translations for languages that are used in censorship-prone locations, such as Arabic.

In addition, we currently have a Portuguese translation submitted by a volunteer to our Trac bug reporting system. We need to get that transferred to Transifex, which is the tool we currently use for translations.

3.5 Improve logging and monitoring

3.5.1 Add monitoring

As part of our larger effort to monitor all of our anti-censorship infrastructure, we need to start monitoring the GetTor service.

3.5.2 Improve log handling

We need to update GetTor's logging to make it more useful for debugging and to collect usage statistics. More specifically, we need to implement log levels to control what information is collected; we need to set up the logrotate utility so that logs are stored in a format that can be easily exported and parsed by other utilities; and we need to extract some simple statistics about usage of GetTor.

3.6 Improve documentation

3.6.1 Specification document

There is an old draft of a specification document that was created several years ago. We need to update it to reflect the current state of GetTor, as well as our near-term plans for the project.

3.6.2 Documentation included with the code COMPLETED

We updated some of the GetTor documentation:

- **README.md** Cleaned up grammar.
- **distribution_methods.txt** Indicated that the XMMP bot needs to be fixed and added several potential distribution methods for future consideration.
- **providers.txt** Added notes on Dropbox and Google Drive. Added Github.

3.7 Improve the code base

3.7.1 Use the Twisted framework COMPLETED

We have refactored the GetTor code to create a Twisted daemon. We have chosen the Twisted framework because it handles all of the necessary networking functionality and includes a good logging system. Using Twisted for GetTor brings the service in line with the BridgeDB service, possibly simplifying maintenance.

Each distribution channel (for example, email or Twitter) consists of one or more services that store or retrieve requests from a database. Another service is responsible for making sure the links GetTor serves point to the latest version of Tor Browser.

3.7.2 Port to Python 3 COMPLETED

GetTor was originally written in Python 2, which goes out of support in 2020. In preparation, we ported GetTor to Python 3.

3.7.3 Implement test functionality

We need to implement unit tests and ensure adequate coverage of the code base.

4 Pluggable Transports: Infrastructure and Maintenance

Pluggable transports [22] help people evade censorship by disguising the traffic between a Tor client, such as Tor browser, and a bridge so that it doesn't look like Tor protocol traffic. This is often necessary in locations where the censor uses deep packet inspection (DPI) to examine traffic as it flows across the network.

Improving the pluggable transport ecosystem is ongoing work with many concurrent lines of research: improving the interface between pluggable transports and Tor; enabling Tor Browser to use other censorship circumvention tools when available; improving each of the currently viable pluggable transports; finding more entities that are willing to support domain fronting in their networks; and maintaining connections with academic research groups so that we stay up to date on emerging work on pluggable transports. In addition to these areas, we must also continue to research and develop the next generation of pluggable transports; Sections 5 and 6 describe that work.

4.1 Improve logging and status messages PARTIALLY COMPLETED

To make it easier for developers to write and maintain high quality pluggable transports and to help us better understand when and how pluggable transports get blocked by censors, we added two new features to support better logging and message passing capabilities for pluggable transports.

The first feature enables pluggable transports to log directly to Tor's log files. Previously, pluggable transports were responsible for their own logging, which meant developers needed to analyze both sets of logs. Trying to determine the order of events across the log files could be error prone. With the release of this feature all of the logs will be in one file, written in a standard form. We are currently defining the format for the log messages a pluggable transport can pass to the Tor process for logging.

The second feature implements a new status handler, making it possible for pluggable transports to report status information back to the Tor control port for use by clients—such as Tor Browser and OONI Probe. This information might help us identify that a particular pluggable transport is being blocked and how the censor is blocking it.

The next step to make use of the status handler is to define key-value pairs in the pluggable transport specification [22] to allow pluggable transports to report useful information.

Other recent related work extended the status messages produced during bootstrap process—the process of connecting to the Tor network and building a circuit—to distinguish between connections to a Tor relay, a pluggable transport, and a simple proxy used to bypass a firewall.

Because a pluggable transport can, itself, use a firewall bypass proxy, we should consider reordering the bootstrap status messages to take this fact into account. A possible order follows:

- `conn_pt` - Connecting to pluggable transport
- `conn_proxy` - Connecting to proxy
- `conn_proxy_done` - Connected to proxy
- `conn_pt_done` - Connected to pluggable transport

4.2 Specify which outbound interface to use

Tor can be configured to use a particular outbound network interface, but there is currently no way to tell a pluggable transport that it should use that interface. We need a mechanism in tor to tell pluggable transports which interface to use. This will also entail work on the pluggable transport side to look for and honor the preference.

4.3 Make pluggable transports more mobile friendly

Currently, when a pluggable transport has been launched but is not being used, it remains running in the background, sometimes even making network connections. In order to use fewer resources and thus be more suitable for use on mobile devices, pluggable transports should become dormant when not in use. We need to analyze the pluggable transports protocol to look for a way to do this.

We also need to assess how dormant mode interacts with message passing between pluggable transports and Tor.

4.4 Better testing for pluggable transports

We need to ensure that we have good test coverage for pluggable transports. This includes getting common pluggable transports included in our Chutney continuous integration testing process. In addition, we need to ensure that new pluggable transports, as well as currently deployed, viable ones, have a high degree of test coverage. We should also encourage Tor developers to alpha-test new and updated pluggable transports so that we can discover issues before release. Finally, we should ensure that we adequately test IPv6 configurations for all current and future pluggable transports.

4.4.1 Reachability self testing

Bridges that offer pluggable transports don't currently have a way to test their reachability. This means that operators may not be able to tell if their bridge is unreachable, for example because of NAT. We need a mechanism to test the reachability of these bridges, particularly those offering obfs4.

The long term solution is to have Tor run the test. A potential design is to have the bridge's Tor client establish a TCP connection with its obfs4 port. Tor can then warn the operator in its log file if the test fails. (We note that this won't solve the problem for any future pluggable transports that run over UDP.)

In the shorter term, while we wait for implementation and deployment, we will create and host a simple web page that asks for an IP address and a port as input. The service then tries to establish a TCP connection to the given tuple, and lets the user know if it succeeded or failed. The service doesn't need to log or remember anything, and we can run it on the host that also runs BridgeDB.

4.5 Enable Tor Browser to use other circumvention tools

Sometimes when access to the Tor network is blocked, other censorship circumvention tools may still work in the censored location. In such a case, Tor Browser might be able to detect the other tool and use it to connect the user to the desired website.

Although these tools generally come with trade-offs related to performance, reliability, and safety, it would be useful to make it easy for Tor Browser to route traffic through them if they are installed. These options could be offered to the user in the pluggable transport selection menu.

One point to consider is that many of these tools attempt to hide their presence to keep censors from finding them, so this work would require cooperation with the makers of the tools.

Notes:

1. Create a list of the censorship circumvention tools that we'd like Tor Browser to use when access to the Tor network is blocked.

2. Contact the makers of these tools and ask if they'd like to collaborate with us by offering an API that Tor Browser could use to route traffic through their systems.

3. Design and implement an update to Tor Launcher so that it can configure Tor Browser to use the tools and add them as options in Tor Browser's pluggable transport selection menu.

4.6 Maintain currently deployed pluggable transports

Although it is important to look ahead to the next generation of pluggable transports (see Sections 5 and 6), we must also attend to the maintenance of currently deployed pluggable transports that we believe work well, such as obfs4proxy and meek [8].

All of our work on pluggable transports requires that we better understand the current censorship landscape. Knowing what works and what doesn't work, in which countries, as well as how censors block Tor, is essential to maintaining current pluggable transports and designing new ones. For more information on understanding how Tor is blocked, see Section 8.

4.6.1 obfs4proxy

Obfs4proxy (or obfs4) is a pluggable transport that defends against active probing attacks so it is resilient against blocking by many censored countries, including, we believe, China. It is also possible that short of blocking obfs4, censors may be degrading its performance.

We need to test if obfs4 is actually being blocked or throttled in China and other censored locations. Our plan to do this follows:

1. Set up several diverse, new, private obfs4 bridges for these tests.

2. Write a client-side script that will test for obfs4 reachability, making sure to measure bandwidth for throttling. COMPLETED

3. Get set up on a virtual private server (VPS) to perform these tests ourselves initially.

4. Contact users in China and other censored locations to run the scripts and send us the date they collect.

If obfs4 is being blocked or its performance degraded, we should figure out how and work to fix it. If it is not, then it is one of the viable pluggable transports, so we need to allocate resources to fixing known issues and maintaining its code base.

4.6.2 Domain fronting

Domain fronting—a system in which a client makes a connection to an uncensored website and that website redirects the connection to a censored website that is in the same domain—has proven resilient to blocking because censors are unwilling to block the uncensored site. This technique has generally been used within the domains of large cloud service providers and requires their cooperation. Meek [8] and Snowflake [19] both make use of domain fronting.

In 2018, reportedly [18] because of political pressure from Russia, Amazon and Google discontinued support for domain fronting. As long as there are cooperating providers, domain fronting is difficult to defeat, so finding other providers who will support it is an important tactic in the fight against censorship.

Other anti-censorship projects that use domain fronting may have lists of companies that are still willing to support the technique. We need to strengthen our collaboration with these projects to foster on-going knowledge sharing.

Encrypted Server Name Indication (ESNI), which is an emerging draft specification for encrypting SNI, the technology that allows a single web server to host multiple websites, may offer some possibilities for domain fronting as it gains wider deployment. We should monitor developments in this area.

4.6.3 meek

As part of ongoing maintenance of meek, we fixed a bug where the meek client for Tor Browser and its child Firefox process would continue to run after Tor Browser was closed.

4.7 Looking to the future

4.7.1 Maintain relationships with research groups

One Tor Project priority is maintaining good relationships with academic research groups that study privacy enhancing technologies in general, and anonymity systems and censorship circumvention in particular. It is essential that we stay current on research on new pluggable transports so that we can continue to help our users maintain access to the free and open internet.

Members of our new anti-censorship team will meet with researchers both by attending conferences and workshops, such as the USENIX Workshop on Free and Open Communications on the Internet and the Privacy Enhancing Technologies Symposium, and by visiting their institutions. We also plan to invite pluggable transport researchers to our biannual Tor meetings.

4.7.2 Consider ways to improve the pluggable transport specification

We want to make it easier for developers and academics to design and implement new pluggable transports and get them easily integrated with Tor so that we can have a well-functioning pluggable transport integration pipeline.

- Assess pain points with the current specification.
- Find out what new features pluggable transport developers would like to see.
- Assess the PT 2.1 [17] draft specification to see where it differs from ours and decide whether we should include some of its features.
- Consider how bridge distribution should factor into the specification. For example, some transports such as meek and snowflake handle bridge information differently than transports whose bridges are distributed through BridgeDB. This results in a different interaction with Tor, and we might consider modifying the spec with the snowflake/broker model in mind.

In general, we should improve our communication with the pluggable transports community to see what they need so that we can get more transports integrated with Tor.

5 Pluggable Transports: Snowflake

While we continue to support and maintain currently deployed pluggable transports, we must also focus our efforts on new ones. Snowflake [19], which builds on Flashproxy [10], is currently the most promising of these next generation pluggable transports.

Snowflake sends traffic through temporary proxies called *snowflakes* that are created within volunteer users' browsers when they visit particular websites. A snowflake provides a connection from a Tor client (such as Tor Browser) to a Tor bridge, enabling the person in the censored location to access the Tor network. A Snowflake client, running with the Tor client, makes a domain-fronted [11] request to the Snowflake *broker*. The broker replies with the IP address and other metadata needed for the client to connect to a snowflake. The client then connects directly to the snowflake using WebRTC, a peer-to-peer protocol implemented in most modern web browsers. The snowflake relays traffic between the Tor client and the Tor bridge. Figure 1 illustrates Snowflake's architecture.

5.1 Document the current broker implementation COMPLETED

An experimental Snowflake prototype is currently integrated in Tor Browser's alpha releases for Mac OS and Linux, but much work remains to make it ready for broad release [20]. As a starting point for necessary design improvements, we have examined and documented the current broker implementation.

https://github.com/ahf/snowflake-notes/blob/master/Broker.markdown

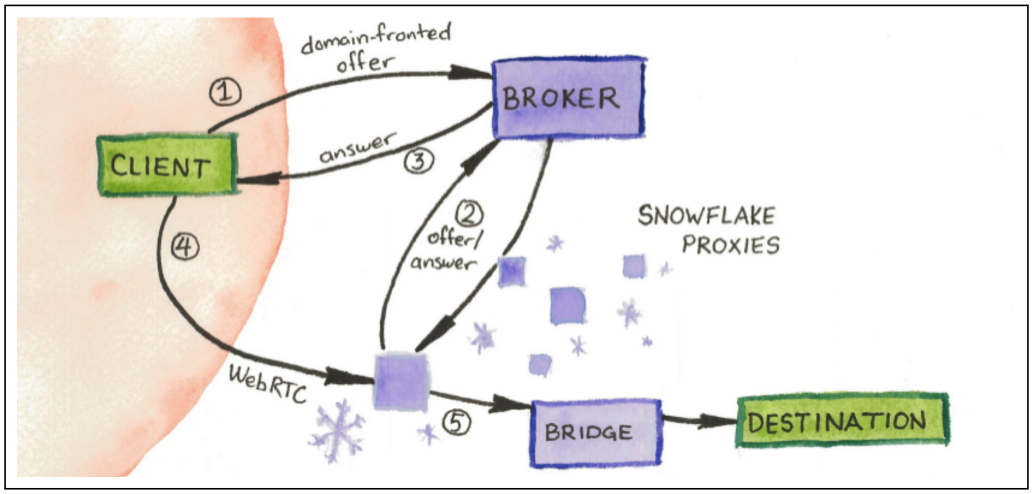

Figure 1: Snowflake architecture [9]

5.2 Improve the Snowflake Protocol

We plan to redesign the Snowflake protocol to make it extensible and to enable us to collect useful metrics. This section outlines ideas and questions for this redesign.

See also: https://github.com/ahf/snowflake-notes/blob/master/Protocol.markdown

5.2.1 Client to snowflake proxy protocol

A Snowflake client connects to a snowflake to send traffic to a Tor bridge and then on to the Tor network. We want to add the ability to send some control messages between the client and snowflake.

Notes:

- Would it be useful for a client to be able to connect to multiple snowflakes? Does this make sense given Tor's current pluggable transport design where the Tor client produces only one output stream?
- Should the client be able to specify, from a set of bridges, which bridge the snowflake should use?

5.2.2 Snowflake to broker protocol

Currently snowflakes do a *long poll* to the broker. A long poll in this context is an HTTP request where the server reads the request, but doesn't send a reply until it has something to say. We should consider, instead, a bi-directional communication link between snowflakes and brokers.

Adding more brokers should alleviate concerns about these links resulting in too many connections for the broker. We could also consider building a back-out mechanism into the broker-snowflake protocol that would allow the broker to signal an overly eager snowflake to wait before attempting to connect again.

Notes:

- Tor relays have an identity key, which allows us to reason about their performance and stability. Should we (optionally?) allow snowflakes to have identity keys? This would allow us to assess snowflakes over time to identify which are more stable than others.
- For extensibility: allow snowflakes to identify which transport they are using, and which version of the transport. This makes it so that we can extend the protocol to handle transports other than WebRTC.
- For extensibility: make sure the snowflake and broker can communicate necessary connection information for transport types other than WebRTC.
- We should consider using WebSocket for messages between snowflakes and the broker. This would give us a versioned, extensible protocol for this communication. It would require a WebSocket handler at the broker and modifying the snowflakes to make WebSocket connections.
- We might have snowflakes advertize a capacity value. This could be a bandwidth value or the number of connections they can handle.

5.2.3 Client to broker protocol

A Snowflake client asks the Snowflake broker to give it a snowflake to relay its traffic through.
Notes:

- Because communication between the client and the broker is domain fronted, which is costly, this protocol should be kept to a minimum.
- We might want to collect statistics about which countries requests are coming from. If we later see more requests than successful connections to snowflakes from a particular location, we can infer that domain fronting works from that location, but WebRTC (for example) is blocked.
- For extensibility: allow clients to request different transport types and transport version numbers.

We want to design this protocol so that it is not Snowflake specific. A client should be able to connect to the broker to request any kind of bridge. Eventually, the broker might be able to replace the BridgeDB service.

We should explore using various options for connecting to the broker:

- IP address: using an IP address rather than a domain name in conjunction with a CDN increases collateral damage by forcing the censor to block everything behind the IP.
- AMP cache fronting: use Google Accelerated Mobile Pages (AMP) caches as a front for the broker.
- DNS tunneling: tunnel requests to the broker over DNS.

5.2.4 Broker to broker protocol

Currently, we have a single Snowflake broker. If that broker were to become unavailable, it would be impossible for snowflakes to announce themselves and for clients to find snowflakes. To make the system resilient to the failure of, or an attack on, the broker, we should consider adding more brokers.

An architecture with multiple brokers would require a protocol for the brokers to reach consensus about which snowflakes are currently available to handle clients. This would likely require each broker to share some metadata about their clients with the other brokers. We should initially make this consensus scheme simple and expand it when we have more experience with the Snowflake system as a whole.

This protocol is lower priority than the others in this section.

5.2.5 Better resilience against Dos attacks for the broker

The broker needs to be more resilient against DoS attacks. In addition to traditional DoS techniques (for example SYN floods, packet floods, memory exhaustion), the broker might be flooded with low-bandwidth snowflakes. An attacker could create a large number of low resource snowflakes, making it likely that these would be distributed to clients, degrading the user experience. We should consider devising a way to test that a snowflake make a reasonable level of bandwidth available before the broker will distribute it to a client.

5.3 Useful metrics

We need to collect some information to better understand where Snowflake users are located, where volunteer snowflakes are located, what kind of load the various components experience, and pain points in the system. Aside from helping us understand the operation of the system, this information would help us detect and respond to censorship events.

- Which countries do connections to snowflakes come from? We use IP geolocation to determine this. COMPLETED
- Which countries do connections to brokers to request snowflakes come from? We use IP geolocation to determine this. COMPLETED
- How much bandwidth use do we see at snowflakes? At bridges?
- How frequently does a snowflake fail to connect to a bridge?
- What are the locations of snowflakes?
- How persistent and reliable are snowflakes?

We would also like to publish real-time information about how many snowflakes are currently registered with the broker and available to serve users. This would help snowflake volunteers understand whether they are needed and whether more snowflakes would currently be useful.

It would also help censored users understand the health of the current snowflake population and potentially help them investigate and solve problems they may have with the system.

In addition, to better understand the address distribution algorithm, we should consider publishing statistics about how frequently a particular snowflake has been given to a user, along with how frequently the broker has decided not to give a user a particular snowflake because it has already been distributed too many times.

https://bugs.torproject.org/21315

5.4 Build Snowflake into Tor Browser

This section describes several areas of work we need to do to create a viable version of Snowflake that we will integrate into the main Tor Browser release.

5.4.1 Reproducible builds

We need to examine Snowflake's dependencies and analyze how difficult it will be to integrate into Tor's build processes.

5.4.2 Include libwebrtc license files in software release

We need to ensure that we include the appropriate Chromium license files for the WebRTC library in our software release. If we decide to use another library, we need to include the appropriate license files for that library.

5.4.3 Add Snowflake to Tor Launcher

We need to add Snowflake to Tor Launcher so that a user can select Snowflake from the pluggable transport menu.

5.5 More Snowflake bridges

The current implementation of Snowflake's client-side proxy hard codes the URL of the Snowflake bridge. To provide more resiliency and to support more load, we need to add more Snowflake bridges. To do this we need to consider what changes will be necessary at the Snowflake components: the client-side proxy, the snowflake, and the broker.

Notes:

- A snowflake might send traffic to any bridge or it may want to specify which bridges it is willing to send to.
- A snowflake might only send traffic to one bridge. If a snowflake sends to multiple bridges, the client needs to have a way to tell the snowflake which bridge it wants to use.
- The broker needs to learn from snowflakes which bridge or bridges they use and from the client which bridge it wants to use. The broker needs to be able to match clients with snowflakes that can forward traffic to an appropriate bridge.

- The client needs to be able to tell the broker which bridge it wants to use. It might also need to tell the snowflake which bridge it wants to use.

In order to support this setup, we need to bring more Snowflake bridges online. We should recruit trusted members of the Tor community to run this first set of Snowflake bridges. The bridges should be capable of a high uptime and be located in areas where censorship is not a big concern.

5.6 Easier ways to run a snowflake

We are in the process of revamping the web page a volunteer visits to make their browser a snowflake. User experience is important here so that the volunteer understands what is required (WebRTC, javascript), and can see when a user is connected to their snowflake.

A drawback to running a snowflake in a browser tab is that browsers try to minimize work when a user is not interacting with a tab. This may become a problem because WebRTC is not generally considered something that needs to run in the background the way, for example, music streaming is.

Another approach is to create a browser extension (or "add-on") to run snowflakes. Installing and enabling the extension would make the browser act as a snowflake. The volunteer wouldn't have to keep a tab open and snowflakes would be more long lived.

Cupcake [4] is software that makes it easy to distribute and run Flashproxy [10], Snowflake's predecessor. Cupcake is distributed either as a Chrome or Firefox add-on or as a module, theme, or app on popular web platforms. The latter causes visitors to the site to become proxies. We are investigating using Cupcake as a browser add-on for snowflakes and UX discussions for this idea have begun. Cupcake has about 4,000 users on Chrome, so offers a promising way to get people to run snowflakes.

With some publicity, we can probably create a community of volunteers who would take pride in helping censored users. We frequently hear from people who want to contribute to the Tor Project but who do not have the resources or skills to run relays. This would be a great way to get those people involved. We should consider ways to gamify running a snowflake to show appreciation and make it more fun.

5.7 Snowflake connection multiplexing

5.7.1 One snowflake supports multiple clients

Currently, each snowflake can handle one client at a time. This is fine if the number of snowflakes is always much bigger than the number of clients. It would be better if each snowflake could handle multiple clients at the same time, and we should implement that now so that we're ready when we have a spike in clients.

5.7.2 One client uses multiple snowflakes

Currently, each client sends traffic through a single snowflake. If that snowflake is slow, performance is degraded. We would like to allow a client to send traffic through multiple snowflakes. This would require implementing a sequencing and reliability layer, possibly with a retransmission mechanism.

A potential privacy benefit is that each snowflake would only see a subset of the client's traffic. A potential downside is that making a number of simultaneous WebRTC connections might help a censor learn that Snowflake is being used.

5.8 Improve safety for users

5.8.1 End-to-end confidentiality for client registration

A Snowflake client uses domain fronting to connect to the broker. The connection to the front domain uses TLS, as does the connection from the front domain to the broker. But the fronting service itself can see the request. It is unavoidable that the fronting service can see the IP address of the client, but we should protect the other metadata that appears in the registration request by encrypting it.

We could use a similar solution to that of Flashproxy [10]. There, the facilitator had a private RSA key and client registration methods were encrypted before being posted to the facilitator. Key material was isolated into a facilitator-reg-daemon process that was separated from the web server and facilitator CGI.

5.8.2 Sanitize Snowflake logs COMPLETED

In certain error situations, for example, when the WebSocket server panics, client IP addresses are written to Snowflake logs. Because this compromises user anonymity, we prevent this from happening. We implemented a log scrubber that uses regular expressions to match IP address patterns so that we can avoid writing them to the logs.

Several Snowflake components produce log files, so we needed to ensure that we fixed this issue for all of them: the client, the server, the broker, and the proxy.

5.8.3 Fingerprinting resistance

In order to defeat DPI, Snowflake's traffic patterns need to be as indistinguishable as possible from regular WebRTC traffic. We need to analyze WebRTC's network layer so that we understand how both DataChannel traffic and MediaStream traffic look and compare that to how Snowflake traffic looks.

5.9 An improved WebRTC library

In addition to data channels for communication between peers, WebRTC comes with a variety of audio and video capabilities that include various codecs. Snowflake only uses the peer-to-peer data communication channels, so building a smaller WebRTC library that does not include the audio/video functionality might be useful. At the same time, we need to analyze whether other WebRTC applications use the library this way; we don't want to make Snowflake easily distinguishable from allowed uses of WebRTC.

A pure Go WebRTC implementation is available from Pions [16], so we might evaluate whether that would meet our needs.

We should also consider using Firefox's WebRTC implementation instead of Chrome's, which would solve the problems with creating a reproducible build on Windows. The drawbacks, however, may outweigh the benefit: using a headless browser is difficult, which is why meek was recently modified to stop doing so; and Tor Browser disables WebRTC to avoid leaks of IP addresses.

5.10 Test environments

5.10.1 Better test coverage

To make sure Snowflake is reliable, we need to ensure that our code has a high degree of test coverage. Part of this will be accomplished by including it in our Chutney continuous integration process.

5.10.2 An automated local test environment COMPLETED

We created a locally networked testing environment for Snowflake that can easily and automatically be set up and run. This includes easy installation and configuration of dependencies. We are using networked docker containers for this environment.

The goal for this environment is to enable us to reproduce bugs that have occurred in the deployed system that we had not been able to reproduce locally.

In the future, it might be interesting to see if we can do some kind of continuous integration with gitlab: `https://docs.gitlab.com/ee/ci/#gitlab-cicd-for-docker`.

5.10.3 Test suite for NAT topologies

One of the strengths of Snowflake (as opposed to Flashproxy) is that WebRTC allows incoming connections to computers that are subject to NAT—a very common situation. To be sure that Snowflake works for censored users in a variety of NAT configurations, we need a test suite that tests against various NAT topologies.

5.11 Add disk space monitoring

We need to set up disk space monitoring at both the broker and the default Snowflake bridge to avoid situations where full disks prevent Snowflake from working. This falls under the umbrella of adding monitoring for all of our ant-censorship infrastructure.

5.12 Address known issues

We have identified several known issues that it is a priority to investigate and address.

5.12.1 snowflake-client needs to stop using my network when I'm not giving it requests

When the user turns Snowflake off, the Snowflake client continues to contact the broker every 10 seconds to ask for a snowflake, resulting in unnecessary network use and thus hurting scalability. The Snowflake client needs to be more defensive by going dormant either after some period during which it has not received a request or whenever it isn't handling a request.

5.12.2 Need something better than client's 'checkForStaleness'

We currently have code in place to check the staleness of a connection between a client and a snowflake to prevent long-lived, broken connections from persisting. But because there is no heartbeat message at this layer of abstraction, pauses in use, such as the user reading a webpage, cause a disconnection after 30 seconds. We need to solve the broken connection problem in a way that does not result in these repeated disconnections.

5.12.3 proxy-go is still deadlocking occasionally COMPLETED

Our test snowflakes were hanging after they had been running for a few days. All of the snowflakes exhibited this behavior, but the more heavily used snowflakes exhibited it first.

Our code incorrectly assumed that the OnIceComplete callback was only called when the connection between the client and the snowflake was complete. That wasn't the case, so the snowflake thought it had a connection with the client when it didn't. Since a proxy-go instance can only handle a certain number of clients at a time, it slowly used up all the available slots.

We implemented two things to fix this. First, we have a channel that one Go routine uses to signal that a connection is complete. Second, if the signal has not been received within a period of time, we do an explicit check to see if the connection is complete.

5.12.4 Guard against large reads COMPLETED

The broker sends and receives relatively small messages when it communicates with the clients and the snowflakes. We have limited the resources we allocate for these messages so that malformed messages don't consume excess memory.

6 Pluggable Transports: Other Next Generation PTs

6.1 Marionette

Marionette, an open source pluggable transport designed and developed by RedJack, LLC, can make Tor traffic look like traffic from a variety of different protocols. Currently, Marionette mainly transforms traffic so that it looks like regular encrypted web traffic. A strength of Marionette is that it can handle a relatively large amount of traffic quickly.

Marionette was originally written in Python. In preparation for integrating it with Tor Browser, it was ported to Go. This port is now ready to be integrated into an experimental version of Tor Browser and the Tor relay code so that a full evaluation can be done. To enable this testing, we have set up a bridge running Marionette.

6.2 HTTPS proxy

HTTPS proxy uses the HTTPS CONNECT method to instruct a proxy server to relay traffic between a Tor client and a Tor relay. The traffic between the client and the proxy is HTTPS traffic, which censors generally don't want to block.

Most simply, HTTPS proxy just passes traffic back and forth between the client and an existing relay, but in this configuration, there is no way to collect metrics about its use. Another way to deploy HTTPS proxy is to include it on a Tor bridge that offers another pluggable transport. The HTTPS proxy handles authentication, and passes the connection request to the other pluggable transport that can collect metrics.

As a measure against active probing attacks, an HTTPS proxy can run a web server alongside the proxy. When the proper credentials are not presented, the web server serves innocuous content.

HTTPS proxy is at an experimental stage, with prototype code submitted to the Tor Project. We need to examine the code and get it ready for deployment and evaluation. In particular, there are a variety of ways that an HTTPS proxy could be identified by its behavior and more research is needed to mitigate them.

6.3 Design and develop new pluggable transports

Censorship circumvention is an arms race, so it is prudent to be looking ahead to the next technique. We need to begin planning, designing, and implementing the new pluggable transports that we'll need if censors learn how to block current ones.

7 Improving User Experience for Poor Networks

To improve the Tor user experience for all of our users we need a better understanding of the factors that contribute to poor performance, especially in locations with poor network infrastructure or mobile-only connectivity. People in censored locations need to take additional steps to connect to the open internet and we need to focus on the user experience of those steps, as well.

7.1 Make the bootstrapping process faster

We need to better understand the points in the bootstrap process that cause delay. Some of the work outlined in Section 4.1 sets us up to collect more information during bootstrapping, including distinguishing whether the connection is being made to a relay, a pluggable transport, or a proxy used to bypass a firewall. Additional planned work will add more granular reporting of events during the bootstrap process, which will allow us to pinpoint the activities that are contributing to delay.

7.2 Better support users on slow networks

There are a variety of tuning parameters, possibly at every level of the Tor code base, that we need to examine with slow networks in mind. Many of these parameters have been set for conditions expected in places like North America or Europe. Users in places that generally have poor networks experience Tor Browser as extremely slow or completely unusable. We need to identify the parameters that might be causing poor performance for people on slow networks and design solutions that will work for all of our users.

7.3 Test pluggable transport latency

OnionPerf [13] measures the performance of the Tor network by downloading files of various sizes from a web server and recording how long each takes to complete. In order to test the effect of pluggable transports on performance, we need to set up a Tor bridge that runs all of the current pluggable transports and configure OnionPerf to perform tests using each pluggable transport on the bridge. Knowing the additional overhead produced by each pluggable transport will help us focus our efforts on improving performance and allow us to make better recommendations to users on slow networks.

7.4 Keep the Tor network healthy

The health of the Tor network is crucial to all of the services Tor provides, including the ability of users to bypass censorship. While it might be ideal to have a team dedicated to keeping the Tor network healthy, most of this work is currently handled by the network development team. This work includes watching for and analyzing anomalous network behavior and helping to resolve network issues as they arise. Specifically, we should do all of the following: establish baselines of expected network behavior; look for and resolve denial of service issues; track connectivity issues between relays; and look for relays approaching the limits of their resources.

8 Understanding Censorship of the Tor Network

To inform our anti-censorship work, we need to better understand the places where Tor is blocked and how censors are blocking it. We need up-to-date information so that we can include good advice to users in Tor Launcher's settings and help them when they ask for support. Knowing historical trends will help us prioritize whether we should develop new pluggable transports or new distribution methods for existing ones.

We currently have a variety of sources of information, but no current comprehensive snapshot of which countries use which methods to block access to the Tor network. We have data that OONI Probe collects, our Tor Metrics timeline [21], anecdotal stories from our users and allies, our censorship wiki, and a timeline published by researchers at UC Berkeley [1]. Some of this information is current; some is older, but may still be accurate. In addition to this information, there is other information that we could be collecting and we describe some of it later in this section.

This comprehensive snapshot should tell us which countries block Tor and how they do it: which ones block Tor directory authorities; which block the public relays; which block the default bridges that are included in Tor Browser; which collect bridge IP addresses from BridgeDB and block them; which use DPI to detect and disconnect pluggable transport connections; which degrade performance for protocols they don't like. When we create this snapshot, we should also put into place a process for keeping it up to date.

8.1 A Tor Browser that detects censorship

In order to get a better understanding of which censored locations specific pluggable transports work in, we need a tool for testing them. Some deployed pluggable transports are quite large, and thus may be unreasonable to integrate into OONI Probe, which is a mobile app. So we would like to make a Tor Browser variant that cycles through the available pluggable transports and reports to the user on which ones work. Of course, we'd also encourage users to share this information with us to aid our work.

A prerequisite for this is better reporting from Tor about the status of the process of bootstrapping a connection to the Tor network. Recent work has defined additional phases in the bootstrap process that allow us to distinguish between connections to relays, pluggable transports, and proxies. Other work enables a pluggable transport to send status and log messages to Tor which can then be made available to Tor Browser and OONI Probe using the Tor control port. These features will provide better information than we have historically had and give us a better idea of what works where.

8.2 Create and maintain an authoritative list of default bridges

Default bridges are the bridges that are pre-configured in Tor Browser. The list of these bridges is currently part of the Tor Browser software repository. This list is used by other projects, notably OONI, but possibly others as well. We want to create an authoritative list of these bridges, maintained by the anti-censorship team, that is easy to parse so that it is usable by other projects.

8.3 Assess status and reachability of default bridges

Default bridges are the set of bridges that are preconfigured in Tor Browser. Currently, the set of bridges included is not updated regularly, and we don't have a good understanding of whether they are still acting as bridges or from which locations they may be censored. We need to implement a process for scanning these bridges to answer these questions.

8.3.1 Default bridge reachability COMPLETED

A default bridge may become unreachable because the operator decides not to run it anymore. We need an automated way of discovering when a default bridge disappears. We have implemented sysmon monitoring of the set of default bridges. The sysmon instance runs every 5 minutes, testing for reachability. If a check fails twice in a row, that is if a bridge is unreachable for 5 minutes, it sends an alert. This falls under the umbrella of adding monitoring for all of our anti-censorship infrastructure.

8.3.2 Default bridge blocking

Default bridges may become unreachable because a censor is blocking them. Currently, in some locations, OONI Probe tests whether these bridges are reachable by checking if the TCP port is open. But this test only assesses whether a bridge is reachable, which doesn't distinguish between a bridge that is offline and one that is blocked. In addition, this test is subject to a false positive if a bridge's TCP port is reachable, but it is blocked using DPI of the actual Tor connection. We need better OONI tests that accurately report whether these bridges work.

As part of this work, we need to expose data about bridge reachability in a form and location that is useful to Tor developers.

We should also look into getting the default bridges tested using reachability tests from external projects, such as Augur [15], Censored Planet [2], or others.

8.4 Understand bridge load and blocking

In order to make good use of Tor bridges, we need to better understand their usage and load, as well as how and when they are blocked by censors. Questions we need to answer include: how are bridges being used; is there a relationship between how BridgeDB distributes them and how much load they have; are they overloaded; are they long or short lived; who runs them and where?

8.5 Get more accurate counts of Tor users

In order to recognize and respond to censorship events—that is, instances when the Tor network is blocked from a particular location—we need accurate counts of the number of people who use the Tor network.

As we detailed in Interim Report 1 [5], when we count Tor users, we don't collect identifying information, but rather estimate the number of users based on the number of requests made to directory authorities for the list of relays in the network. Using this methodology, we estimate that we have approximately 2 million Tor users a day.

Counting this way can be especially inaccurate in some censored countries. For example, in some locations, you can successfully request the Tor consensus from a directory authority, but when you attempt to make a connection to a guard relay, the censor uses deep packet inspection (DPI) to detect the Tor protocol and block the connection. In a situation like this, we count the directory authority request as a user, but the client clearly was not able to actually use Tor. This leads to inflated user counts exactly when a censor blocks Tor.

Recent work [12] that uses newer privacy-preserving data collection, produced an estimate of about 8 million Tor users in a day. We need to validate that research and consider whether we should implement its methodology into our collection of user statistics.

9 Making Tor Bridges Easier to Use

9.1 Help NGOs use private bridges

While our eventual goal is to create automated methods for bridge address distribution that are robust against all censors, in the short term, we need to improve current methods. Many NGOs would be happy to run private bridges and distribute their addresses to their members and allies. In particular, groups working in China would like to run bridges with the obfs4 pluggable transport. To make this feasible, we need to create usable documentation to help them set up bridges and distribute the addresses to their users. To test our processes and get user feedback, we should walk a few NGOs through the set-up and distribution process.

A second phase of supporting the use of private bridges by NGOS will entail documenting how to run a collection of bridges. An improvement to consider is making it possible to run multiple bridges on a single Tor instance. As well, we should make it possible for NGOs to run their own instances of BridgeDB.

Research to understand whether private bridges actually work in China is also needed. We need to run recurring experiments from within China to test the following: whether obfs4 bridges are reachable; whether the bootstrap process can complete; and the throughput that is possible—that is, whether connections to private bridges are being degraded by censors.

Throughout this work, we will collect information that will help us devise best practices for how many private bridges are needed per user and where it is best to host the private bridges.

9.2 Improve automated bridge set-up

The introduction of the moat feature in Tor Browser 8.0 significantly simplifies bridge set-up, but it would be better if the average user didn't need to know about bridges at all. Ideally, a user would start Tor Browser and it would try to connect to the Tor network. If connecting to a publicly-listed relay failed, it would try to connect to a bridge, if that failed, it would try bridges offering various pluggable transports. Throughout this process, it would keep the user informed of progress. It could save the connection method that worked to be used for the next time Tor Browser starts.

10 Expanding Community Outreach

To help people bypass censorship using Tor, we need to collaborate better with users in censored locations. We have contacts on the ground in many countries, both individuals and NGOs. We should make a list of who we know and where they are located and contact them to find out the current situations in their countries. In addition, OONI's Partnership Program [14] includes 25 digital rights organizations, many of them in censored countries. We need to leverage these connections to learn more about whether Tor is blocked in various places around the world and what mechanisms are used to block it. We can then collaborate on ways to get Tor working again. We can prioritize countries where we believe Tor to be blocked in some way. We are particularly interested in better understanding the state of internet censorship in China. We also need a plan for contacting our partners regularly so that our knowledge stays current.

In addition to censorship, these collaborations can help us better understand performance and usability issues that people may experience while using Tor. Our current global south initiative, where we have been visiting users and conducting usability testing, provides a model for how we want to engage with allies and partners world wide.

11 Tails

Tails—the amnesic incognito live system—is an operating system that aims to protect users' privacy and anonymity and help them evade censorship. Tails runs from a USB stick or DVD, independent of the operating system of the computer it runs on. Tails does not store any information on the computer's hard disk, so after the computer is restarted, there is no evidence that Tails was used and no trace of what it was used for.

Tails comes with a variety of applications pre-configured to help keep users safe, including Tor Browser, an instant messaging client, and an email client. All of the applications on Tails are configured to connect to the internet over the Tor network.

11.1 Address known clock-related issues

When the hardware clock of the computer Tails is running on is not set to the current, correct UTC time, it is very difficult for applications to connect to the Tor network using regular bridges or bridges with pluggable transports.

Part of the problem will be mitigated by plans for Tor to allow connections when the client has a slightly older view of the Tor consensus, but the larger issue is that Tails can't tell if its host computer's clock time is expressed in local time or UTC.

The Tails team has a good design ready to fix this issue. Parts of the plan will have the added benefit of making the Tails synchronization system safer for all Tails users. The Tails time UX will be improved by displaying time in the local time zone on the desktop.

11.2 Cache the Tor consensus

Many users in censored locations have slow internet connections. To improve usability, Tails plans to cache Tor consensus information, so that it can be fetched less frequently, which will improve usability.

11.3 Improve network connection UX

In the context of an operating system such as Tails, whether or not Tor is blocked is only part of the story: users also encounter captive portals, networks that block spoofed MAC addresses, and so on. So to make the flow from starting Tails to being connected to the Tor network seamless, a more global approach is needed.

The Tails team has a design that has already undergone usability testing with paper prototypes.

11.4 Make bridges easier to use

11.4.1 Better bridge configuration

Currently, Tails does not include default bridges for applications other than Tor Browser. This means that users in censored locations need to find and configure their own bridges before they can begin to use Tails. The Tails team plans to begin including default bridges. In addition, a mechanism for persistently configuring non-default bridges and pluggable transports is planned.

11.4.2 Moat

The Tails team plans to integrate automatic bridge and pluggable transport configuration using Tor Browser's moat feature. Implementing this depends on devising a plan to allow a GUI application to access the internet without using the Tor Network.

11.5 Support more pluggable transports

11.5.1 meek

Because the meek pluggable transport (see Section 4.6.2) has proven resilient to blocking in some of the more challenging censored locations, the Tails team wants to include support for it. However, in order to avoid over-taxing meek bridges, the work described in Section 11.4.1 should be completed first, so that other types of bridges are available for locations where meek is unnecessary.

Like the plan for supporting moat, supporting meek will require a GUI application that connects to the internet without using Tor.

11.5.2 Snowflake

After Snowflake is offered in the stable version of Tor Browser, the Tails team plans to offer Snowflake support as well.

12 Conclusion

Today, free and open communication on the internet is essential for the growth and maintenance of strong communities and societies. Threats to this open communication come from a variety of state and non-state censors. The Tor network is a robust and highly scaled communications network that protects communications metadata, and as such has an important role to play in combatting these denials of service.

In this report, we detail our plans for making significant progress in the following areas: improving the BridgeDB service to make it more usable and resilient; improving the GetTor service to better distribute Tor software to users in censored locations; improving existing pluggable transports and designing and developing new ones; improving the user experience for people with poor network connections; better understanding censorship of the Tor network so that we can help users evade it; making it easier for people in censored locations to use Tor bridges; and expanding our community outreach to help people in more censored locations. We also briefly describe some of the anti-censorship work planned by Tails, an affiliated project.

13 Acknowledgements

This research was supported by NSF grants CNS-1526306 and CNS-1619454, and by DARPA contract HR0011-16-C-0056. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF or DARPA.

Some of the content of this report was taken from the Tor Project's public bug tracking system.

We thank the Tails project for providing information about their plans for anti-censorship work.

References

[1] Sadia Afroz and David Fifield. Timeline of tor censorship. https://www.icsi.berkeley.edu/~sadia/tor_timeline.pdf.

[2] Censored Planet. https://censoredplanet.org/.

[3] CollecTor. https://metrics.torproject.org/collector.html.

[4] Cupcake. https://github.com/glamrock/cupcake.

[5] Antonela Debiasi, Roger Dingledine, Arthur Edelstein, Alexander Færøy, Matthew Finkel, David Goulet, Kat Hanna, Maggie Haughey, George Kadianakis, Iain R. Learmonth, Alison Macrina, and Maria Xynou. Addressing denial of service attacks on free and open communication on the internet. Technical Report 2018-11-001, The Tor Project, November 2018.

[6] Frederick Douglas, Rorshach, Weiyang Pan, and Matthew Caesar. Salmon: Robust proxy distribution for censorship circumvention. *Proceedings on Privacy Enhancing Technologies*, 2016(4), October 2016.

[7] Arun Dunna, Ciarán O'Brien, and Phillipa Gill. Analyzing china's blocking of unpublished tor bridges. In *8th USENIX Workshop on Free and Open Communications on the Internet (FOCI 18)*, Baltimore, MD, 2018. USENIX Association.

[8] David Fifield. meek. https://trac.torproject.org/projects/tor/wiki/doc/meek.

[9] David Fifield. *Threat modeling and circumvention of Internet censorship*. PhD thesis, University of California, Berkeley, December 2017.

[10] David Fifield, Nate Hardison, Jonathan Ellithorpe, Emily Stark, Roger Dingledine, Phil Porras, and Dan Boneh. Evading censorship with browser-based proxies. In *Proceedings of the 12th Privacy Enhancing Technologies Symposium (PETS 2012)*. Springer, July 2012.

[11] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies*, 2015(2):46–64, 2015.

[12] Akshaya Mani, T. Wilson-Brown, Rob Jansen, Aaron Johnson, and Micah Sherr. Understanding tor usage with privacy-preserving measurement. In *Proceedings of the Internet Measurement Conference 2018, IMC 2018, Boston, MA, USA, October 31 - November 02, 2018*, pages 175–187, 2018.

[13] OnionPerf.
https://github.com/robgjansen/onionperf.

[14] OONI Partnership Program. https://ooni.torproject.org/get-involved/partnership-program/.

[15] Paul Pearce, Roya Ensafi, Frank Li, Nick Feamster, and Vern Paxson. Augur: Internet-wide detection of connectivity disruptions. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 427–443, 2017.

[16] Pions. https://github.com/pions.

[17] Pluggable Transport Specification Document. https://github.com/Pluggable-Transports/Pluggable-Transports-spec.

[18] Bruce Schneier. Censorship in the Age of Large Cloud Providers.
https://www.schneier.com/essays/archives/2018/06/censorship_in_the_ag.html.

[19] Snowflake. https://trac.torproject.org/projects/tor/wiki/doc/Snowflake.

[20] snowflake Transport Evaluation (Preliminary). https://trac.torproject.org/projects/tor/wiki/doc/PluggableTransports/SnowFlakeEvaluation.

[21] Tor Metrics News.
https://metrics.torproject.org/news.html.

[22] Tor pluggable transport specification. https://spec.torproject.org/pt-spec.

[23] 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.