

Troll Patrol: Detecting Blocked Tor Bridges

by

Ivy Vecna

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2024

© Ivy Vecna 2024

Some rights reserved.



Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

License

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this licence, visit <https://creativecommons.org/licenses/by/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Abstract

Tor is an important tool for protecting people against Internet surveillance and censorship. Therefore, some governments that wish to monitor or restrict their people's use of the Internet attempt to block access to Tor. Bridges are circumvention proxies that provide routes around this censorship, enabling people to access Tor, even in countries that ordinarily censor it. However, a motivated censor may work to identify these bridges and block access to them.

To impede the censor's attempts at identifying and blocking bridges, reputation-based systems for bridge distribution such as Hyphae, Salmon, and Lox have been proposed. These systems place greater trust in users when the bridges they know remain uncensored and reduced trust in users when bridges they know become censored. In order to enact these changes in trust, it is necessary to know which bridges have been blocked and which have not, but Tor does not currently have a systematic way to detect blocked bridges.

In this work, we present Troll Patrol, a system for automatically detecting censorship of Tor bridges. This system infers bridge reachability based on already-existing bridge usage statistics and novel anonymous user reports that we design for this purpose. We evaluate our system using a simulation and demonstrate that user reports improve our ability to detect bridge censorship, compared to using statistics on bridge use alone. We describe an attack that allows the censor to evade detection if classification of bridge blockage relies on bridge statistics alone, and we demonstrate that user reports allow us to defend against this attack.

Acknowledgments

I am incredibly grateful to my supervisor Ian Goldberg for accepting me as one of his students and providing so much support, guidance, and expertise. I look forward to our continued work together as I pursue my PhD. I would also like to express my deepest gratitude to Lindsey Tulloch, who suggested this project to me and was willing to answer my plethora of questions about Lox, and to Simon Oya, who took the time to help me when I was struggling with the analysis section and patiently walked me through many simple things I found confusing. I thank Diogo Barradas and Urs Hengartner for generously agreeing to be the readers of my thesis and for their feedback.

I am thankful to Cecylia Bocovich for providing insights about Tor and bridges, Diogo Barradas, who taught me much of what I know about the field of censorship circumvention, Adrian Cruzat La Rosa, who helped me with some of the early concepts, Guy Coccimiglio, who worked with me on an early version of this project when we took a class together, and Thomas Humphries and Lucas Fenaux for talking through some analysis ideas with me.

Finally, I would like to thank everyone in CrySP who made me feel at home in the lab.

This work benefitted from the use of the CrySP RIPPLE Facility at the University of Waterloo.

Table of Contents

Author's Declaration	ii
Abstract	iii
Acknowledgments	iv
List of Figures	ix
List of Tables	x
1 Introduction	1
2 Background	7
2.1 Lox	7
2.1.1 Lox Credentials	8
2.1.2 Using Lox	10
2.1.3 Other Parameters	12
2.2 Measuring Censorship	12
2.2.1 Direct Scans	13
2.2.2 Reverse Scans	14
2.2.3 Indirect Scans	15
2.2.4 Bridge Statistics	16

2.2.5	User Reports	17
2.3	Conclusion	18
3	System Design	19
3.1	Threat Model	20
3.1.1	Rdsys	20
3.1.2	Lox Authority	21
3.1.3	Troll Patrol	21
3.2	Negative Reports	22
3.2.1	Proof of Bridge Knowledge	23
3.2.2	Delivery	23
3.2.3	Processing	25
3.3	Positive Reports	25
3.3.1	Bridge Token	26
3.3.2	Positive Report Proof	28
3.3.3	Delivery and Processing	31
3.4	Analysis	32
3.4.1	Possible Approaches	32
3.4.2	Analysis for Lox Bridges	33
3.4.3	Example Analysis	35
3.5	Conclusion	39
4	Simulation	40
4.1	Rdsys	41
4.2	Lox Distributor	41
4.3	Bridges	42
4.4	Extra-Infos Server	42
4.5	Censor	43

4.5.1	Speed	43
4.5.2	Secrecy	44
4.5.3	Totality	45
4.5.4	Bootstrapping Period	45
4.6	Users	47
4.6.1	Censor Agents	47
4.6.2	Honest Users	47
4.7	Troll Patrol	50
4.8	Simulation Operation	50
4.9	Optimizations	51
4.9.1	Disabling Open Invitations	51
4.9.2	Reducing the Number of Censor Agents	52
4.10	Conclusion	53
5	Evaluation	54
5.1	Experiment Setup	54
5.2	Results	56
5.3	Discussion	57
5.4	Conclusion	64
6	Limitations and Future Work	65
6.1	Limitations	65
6.1.1	Country-Level Assumptions	65
6.1.2	No Model of Social Graphs	66
6.2	Future Work	66
6.2.1	Detecting Mass-Blockage Events	66
6.2.2	Shorter Analysis Periods	67
6.2.3	Report Submission	67

6.2.4	Negative Report Encryption Key Distribution	69
6.2.5	Limiting Negative Reports	69
6.2.6	Active Scans	71
6.2.7	Evaluation with Additional Types of Censors	71
7	Conclusion	72
	References	73
	APPENDICES	82
A	Source Code Availability	83

List of Figures

2.1	Level migrations that are possible in Lox	12
3.1	The structure of a negative report	22
3.2	The structure of a proof of bridge knowledge	23
3.3	The structure of a positive report	26
3.4	The structure of a bridge token	27
3.5	Example analysis for stage one	37
3.6	Example analysis for stage two	38
5.1	Accuracy by user report submission probability	58
5.2	Accuracy by harshness	59

List of Tables

1.1	Pluggable transports used in Tor	3
5.1	Experimental configurations	56
5.2	Results of experiment 1 with overt censor	60
5.3	Results of experiment 1 with flooding censor	61
5.4	Results of experiment 2 with overt censor	62
5.5	Results of experiment 2 with flooding censor	62

Chapter 1

Introduction

Tor [DMS04] is an important tool for anonymity on the Internet. Tor users wrap their traffic in layers of encryption and route it through multiple relays called *nodes* to obfuscate its path. A user's Tor client learns the entire list of available nodes by downloading the Tor consensus [TP24e], and it selects three nodes from this list: an entry (often called *guard*) node, a middle node, and an exit node. The client negotiates encryption keys with each node and encrypts its traffic first for the exit node, then for the middle node, and finally for the entry node. The client sends this thrice-encrypted traffic to the entry node, which removes a layer of the encryption and forwards the now twice-encrypted traffic to the middle node. The middle node similarly removes a layer of encryption and forwards the traffic to the exit node. Only at the exit node is the last layer of encryption removed, revealing the decrypted version of the user's request. (Similarly, when the destination responds, each node adds a layer of encryption.) In this way, Tor provides privacy even from the operators of the network, following the Decoupling Principle [SIWR22]: the entry node can identify the user but not the destination or contents of their traffic, the exit node knows the destination (and possibly contents) of the traffic but not the identity of the user, and the middle node learns no sensitive information, only that it is relaying encrypted traffic to and from other Tor nodes.

Tor was not originally intended as an anti-censorship tool, but it is very useful for this purpose. (This is a natural extension of privacy and anonymity: discriminately blocking certain activities and punishing dissident speech is a much greater challenge to a censor that cannot determine people's activities or identify dissident speakers.) Thus, an entity that wishes to restrict or monitor people's use of the Internet may work to block access to Tor; major blockages of the network have been observed in countries such as China [WL12], Iran [pho11], and Russia [ggu21], just to name a few. The most basic approach to blocking Tor is trivial. The identities of Tor nodes are public; this censor can simply download the Tor consensus and block access

to the IP addresses (or IP:port pairs) users would use to connect to those nodes. To combat this attack, users may use anti-censorship proxies called *bridges* [DM06] to gain entry into the Tor network.

The simplest form of a bridge is just an unlisted Tor entry node, but these are not difficult to block. Tor does not normally attempt to resist traffic profiling [MTC17, TP17], so some censors use techniques such as deep packet inspection (DPI) to identify and block Tor traffic [TP11b, Run12, pho12a, pho12b]. If bridges run vanilla Tor (i.e., the Tor protocol without modifications), they may be identified using these techniques [MTC17]. A number of works attempt to resist DPI using either steganographic protocols [WWY⁺12, MLDG12, HNGJ16, BSR17, BSRN20, FBS22], which hide Tor traffic inside other protocols or attempt to replicate other protocols, or by using fully encrypted protocols [Sc24, Lei10, FJ23, TP24i, WPF13], which aim to be difficult to fingerprint. (If a censor attempts to block something it cannot fingerprint well, it risks overblocking.) In the Tor context, these circumvention protocols are called *pluggable transports* (PTs) [TP24k]. Tor currently uses four pluggable transports: meek, Snowflake, lyrebird, and WebTunnel [TP24c]. We briefly describe these below and summarize them in Table 1.1.

meek Meek uses a technique known as domain fronting [Dav15] to make it difficult for a censor to differentiate between inconspicuous HTTPS traffic to a major content delivery network and meek traffic. The approach of meek is not to hide the bridge’s existence and location from the censor but to tie the bridge so tightly to essential infrastructure that the censor cannot block the bridge without also suffering significant disruption of regular Internet use.

Snowflake Snowflake [BBF⁺24] iterates on a concept proposed by Flashproxy [FHE⁺12], which argues that if many volunteers around the world allow their web browsers to act as short-lived circumvention proxies, it would be very difficult for the censor to block them effectively. Proxies using the Snowflake PT (called “snowflakes”) are ephemeral. Users learn about them from a broker that is accessible using domain fronting or similar techniques. Users connect to these snowflakes using WebRTC, a common protocol used in video calls. (However, Snowflake uses data channels, rather than the media streams intended for WebRTC audio and video calls; future work may be required to improve Snowflake’s resistance to protocol fingerprinting [BBF⁺24].) The approach of Snowflake is not to rely on long-term secrecy of the proxies but to incur high enough proxy churn to overwhelm the censor’s efforts to identify and block all the snowflakes. When one snowflake is blocked, users simply move on to another.

lyrebird Lyrebird [TP24i] (formerly called “obfs4” and still referenced as such in many places) is a fully encrypted protocol. Lyrebird bridges are easy to block if the censor can identify them

Table 1.1: Pluggable transports (PTs) used in Tor. PTs that require keeping the bridge secret from the censor are indicated in bold text; we focus on bridges that use these PTs. While Snowflake uses WebRTC, future work may be required to prevent censors from distinguishing Snowflake traffic from WebRTC video calls.

PT	Steganographic or fully encrypted	Resistance to censorship
meeq	Steganographic (HTTPS)	Expensive to block
Snowflake	(Partially) Steganographic (WebRTC)	High bridge churn
lyrebird	Fully encrypted	Bridge is kept secret
WebTunnel	Steganographic (HTTPS)	Bridge is kept secret (may be expensive to block)

as such. They are not ephemeral like snowflakes. They are easily isolated from other traffic (by having a unique IP address and port), so they are not expensive to block once identified. Thus, it is necessary to prevent censors from identifying lyrebird bridges.

WebTunnel WebTunnel [sg24] (based on HTTPPT [FW20]) is a new pluggable transport that uses WebSocket-like HTTPS traffic to carry Tor traffic. WebTunnel runs on existing web servers; only users who know a secret path (sent encrypted with TLS) can connect to the proxy service. Because WebTunnel uses or resembles normally allowed protocols (HTTPS, WebSocket) and runs on the same IP address and port as an actual web server, it may be more expensive to block. However, if the website is not essential, the censor may be willing to block it along with the WebTunnel bridge. Thus, it may still be necessary to prevent censors from identifying WebTunnel bridges.

As discussed above, some bridges need to be made available to users but kept secret from censors. To prevent easy identification by censors, these bridges are not listed in the Tor consensus, and they use PTs to hide the nature of the traffic. However, a number of techniques may still allow censors to identify possible bridges [Din11b]. Censors can and do actively probe suspected servers to see if they act like Tor bridges [EFW⁺15, twi12]. To defend against active probing, bridges can refuse to respond, or respond normally with another expected service [FW20, sg24], unless the client demonstrates knowledge of some secret [TP24i, SJP⁺11, WPF13]. Probing resistance is an essential part of modern PTs.

While probing-resistant PTs aim to protect bridges from discovery by censors, censors may still be able to use the same avenues that regular people use to learn about bridges. Tor represents bridges as *bridge lines*. A bridge line lists the information needed to connect to a bridge (such as its PT, IP address and port, unique fingerprint, cryptographic keys, and secret to defend against

active probing). Bridge lines for secret bridges should not be shared publicly; the information used by people to connect to a bridge can also be used by censors to identify the bridge, confirm that it is a bridge, and block it. A public *hashed fingerprint* can be computed from the private fingerprint listed in the bridge line, and this hashed fingerprint can be used publicly to identify the bridge without enabling censors to block it.

The Tor Project holds a set of bridges it has been entrusted with distributing. There must be some way to distribute bridge lines representing these bridges to genuine users, without also exposing them to censors. Historically, the backend for distributing these bridge lines has been BridgeDB [TP24a]. BridgeDB is currently being replaced by rdsys, the *resource distribution system* [TP24i]. The distribution backend partitions the set of bridges it knows and provides different partitions to different distributors [TP24b]. Currently, the publicly accessible distributors are HTTPS, email, moat [TP24j] (an interface within Tor Browser that uses domain fronting), and Telegram. In an attempt to prevent bridge enumeration, the number of bridges distributed to each requesting user is limited. For example, if the same IP or email address requests bridges multiple times during the same time period, the same bridges are returned. IP addresses from the same /24 block are treated as the same IP address, and emails are only accepted from certain email providers (currently Riseup and Gmail). However, these protections have long been known to be insufficient against a motivated censor. China was able to enumerate HTTPS-distributed bridges in 2009 by requesting these bridges from many different IP subnets, and it was also able to defeat the email defense in 2010 [Din11b]. Bridge enumeration through repeated queries of these channels appears to have remained a capability of censors since [Boc21b, TP20].

In response to these bridge enumeration attacks, new reputation-based systems for distributing Tor bridges and other circumvention proxies have been proposed. These systems allow trusted users to invite friends they trust, and they reward honest users while punishing users who appear to collude with censors. Proximax [MML11] has the goal of maximizing the number of overall user-hours of access to proxies. Trusted users within the system are given individualized host names that allow access to a set of proxies. These users share their proxies however they wish with friends (but these friends do not become users of the Proximax system itself). Information is tracked about how many user-hours of access those users' proxies have provided, as well as the risk of the users inviting a censor. This information determines who should distribute new proxies. Invitations of new trusted users to the system (as opposed to recipients of information about the proxies) are rare, and which users should be allowed to invite friends is determined by both their own performance and their friends' performance.

rBridge [WLBH13] focuses on protecting privacy while leveraging trust networks. Users earn reputation credits based on their known bridges' uptime, and when their bridges become blocked, they can use these credits to replace them with new bridges. Users with enough reputation credits gain the ability to invite friends to join and learn about bridges. rBridge uses

an anonymous credential scheme to track user reputation without compromising privacy. Hyphae [LdV17] adopts the anonymous credential scheme from Chase et al. [CMZ14] to provide better efficiency compared to rBridge.

Rather than award reputation credits, Salmon [DRPC16] assigns users discrete trust levels, where users' trust levels go up if their proxies remain unblocked or down if their proxies become blocked. Trusted users can invite friends, and Salmon attempts to assign those new users to the same proxies as the friends who invited them. Salmon tracks each user's trust level, their suspicion level (i.e., the probability that the user is working for a censor), and their recommendation graph, associating users with their invited friends. Unlike previously described reputation systems that require an invitation from an existing user to join, Salmon can optionally allow new users to join without invitation. However, it requires that new users joining in this way use well-established Facebook accounts to do so, thus relying on Facebook's identification and removal of fake accounts to limit bridge enumeration.

Lox [Tul22, TG23] combines a discrete trust level system as in Salmon with the anonymous credential scheme of Chase et al. as in Hyphae. If a highly trusted user's bridges become blocked, that user can migrate to different bridges, but they lose trust levels in the process. If they migrate too many times, they lose the ability to regain trust entirely. To prevent a censor from learning many bridges by creating many trusted "friend" accounts, invited Lox users join the same bridges as their friends. Additionally, invited users inherit the blockage migration count from the friends who invited them. Like Salmon, Lox aims to allow users to join without knowing an existing trusted user, but it eschews Salmon's Facebook option, envisioning instead that new users can join using techniques such as those currently available (e.g., email or HTTPS).

The Tor Project is currently integrating Lox into Tor [TP24h], so we consider Lox our primary exemplar in this work. In order for Lox or another reputation-based bridge distribution system to penalize misbehaving users and reward honest ones, it is necessary for the system to have insight into which bridges are accessible to users and which are not. Currently this problem has not been adequately solved. The Tor Project's primary source of data on bridge reachability is approximate per-country counts of unique IP addresses from which bridges received connections each day. These are reported by bridges and made publicly available [TP24d]. These statistics are useful, but we find that they are insufficient on their own. In particular, these connection counts can be artificially increased by a censor that blocks regular users from connecting to a bridge but makes its own connections from various IP addresses it controls in order to avoid detection.

By introducing a privacy-preserving user reporting system alongside these extant bridge statistics, we show that we can improve the rate of censorship detection when the censor does not perform the described attack, as well as defend against this attack, a feat that is not possible

using bridge statistics alone. We demonstrate these claims in a simulation, and we find that if 25% of users submit reports, we can detect over 85% of blocked bridges in spite of this attack. Our thesis statement is as follows:

It is possible to design a bridge blockage detection system that harnesses privacy-preserving user reports in conjunction with bridge usage statistics to successfully detect more blocked bridges than would be detected using only the bridge statistics. These user reports can allow such a system to detect censorship even when the censor takes measures to hide it.

Our contributions are as follows:

- We design Troll Patrol, a system for automatically detecting censorship of Tor bridges. A primary component of Troll Patrol is an anonymous reporting system. Reports may be submitted only by users who meet certain criteria (such as being a legitimate user of a bridge or being a highly trusted Lox user), but we ensure these criteria without identifying users.
- We provide a reference implementation of our design in Rust and supply modifications to the existing Lox implementation to support integration with Troll Patrol.
- We design and implement a simulation for Lox and Troll Patrol and use it to evaluate Troll Patrol. Noting that one stated limitation of Lox was unoptimized parameters [Tul22, TG23], we believe that our simulation may be independently useful for testing Lox in various configurations (e.g., with different processes for deciding which users learn which bridges).

The remainder of this thesis is organized as follows. Chapter 2 discusses relevant background on Lox and techniques for measuring censorship. Chapter 3 describes our threat model and the design of our system to defend against those threats. Chapter 4 details the design of a simulation we develop to evaluate our system, and Chapter 5 presents the results of our evaluation using this simulation. We resolve our work with a discussion of limitations and future work in Chapter 6 and conclude in Chapter 7.

Chapter 2

Background

This chapter describes the design of Lox and considerations for measuring censorship.

2.1 Lox

We begin with a description of the Lox bridge distribution system, as described in the Lox thesis [Tul22] and paper [TG23] and implemented in the Tor Project’s Lox codebase [TP24h].

Lox consists logically of two servers, the Lox Authority (LA), which is the primary Lox server, and a Lox invitation token distributor (named `BridgeDb` in the Lox code but distinct from BridgeDB, the predecessor to rdsys). These two components are currently combined into the same server, called the Lox Distributor, but they could be separate entities. Lox is designed to work with rdsys. Rdsys maintains a database of bridges and shares some subset of these bridges with the LA to distribute to users. When the LA learns about bridges from rdsys, it adds their bridge lines to its own database and partitions this database into open-entry and invite-only buckets. Open-entry buckets contain one bridge each and are available to new Lox users who are not invited by existing trusted users. Invite-only buckets contain three bridges and are only available to new Lox users if those users are invited by existing users. The LA also issues anonymous Lox credentials to users, which allow them to learn their bridges and perform Lox protocols.

We first describe Lox credentials, then provide a brief overview of the ways users can use Lox.

2.1.1 Lox Credentials

Lox uses the MAC_{GGM} -based anonymous credential scheme from Chase et al. [CMZ14]. The Lox Authority acts as both issuer and verifier. Let A and B be generators of a fixed group \mathbb{G} of prime order ℓ , written additively, such that $\log_B(A)$ is unknown. The Lox Authority's secret key is the vector

$$(\tilde{x}_0, x_0, x_1, \dots, x_n) \xleftarrow{\$} (\mathbb{Z}/\ell\mathbb{Z})^{n+2}$$

where n is the number of attributes in the Lox credential (currently 6). The corresponding public key is the vector

$$(X_0, X_1, \dots, X_n) = (\tilde{x}_0A + x_0B, x_1A, \dots, x_nA).$$

A Lox credential consists of a MAC (P, Q) and a vector of attributes $(m_1, \dots, m_n) \in (\mathbb{Z}/\ell\mathbb{Z})^n$, where

$$b \xleftarrow{\$} (\mathbb{Z}/\ell\mathbb{Z})^*$$

$$P = bB$$

$$Q = \left(x_0 + \sum_{i=1}^n x_i m_i \right) P.$$

Lox credentials have the following attributes:

- Φ : A random unique ID
- t : The date of the user's last trust level change
- L : The user's trust level (from 0 to 4, where level 0 is untrusted and levels 1–4 are trusted)
- β : The user's bucket, consisting of (i, K_i) where i is a bucket ID that can be used to retrieve the encrypted bucket from a table of encrypted buckets, and K_i is a symmetric key that the user can use to decrypt the encrypted bucket
- a : The number of available invitations with which the user can invite friends
- d : The number of bucket blockages the user has observed (i.e., the number of times the user has performed a blockage migration)

Where Lox refers to an attribute by a symbol (e.g., Φ), we will refer to that attribute's value in equations by m with the symbol as a subscript (e.g., m_Φ) to emphasize that the attribute is a scalar. Users can show their credentials to prove that they have certain attributes. The MAC is rerandomized before credential shows:

$$t \xleftarrow{\$} (\mathbb{Z}/\ell\mathbb{Z})^*$$

$$(P', Q') = (tP, tQ)$$

The simplest way to show one's credential would be to present all the credential's attributes along with the MAC. The LA could then verify this MAC on the given attributes by simply checking

$$Q \stackrel{?}{=} (x_0 + \sum_{i=1}^n x_i m_i)P.$$

However, this requires all attributes and the MAC (P, Q) to be revealed. Instead, it is desirable to hide some or all of the credential's attributes as well as Q . To blind an attribute, the user forms a Pedersen commitment to that attribute:

$$z_i \xleftarrow{\$} (\mathbb{Z}/\ell\mathbb{Z})^*$$

$$C_i \leftarrow m_i P + z_i A.$$

The user also forms a Pedersen commitment to Q :

$$z_Q \xleftarrow{\$} (\mathbb{Z}/\ell\mathbb{Z})^*$$

$$C_Q \leftarrow Q + z_Q A$$

The user computes

$$V \leftarrow (\sum z_i X_i) - z_Q A, \text{ for } i \text{ where } m_i \text{ is blinded}$$

and uses a Schnorr proof to prove in zero knowledge that

1. For each blinded attribute m_i , C_i is a valid commitment to m_i
2. $V = (\sum z_i X_i) - z_Q A$, for i where m_i is blinded

along with any additional statements.

The user submits P , any revealed attributes m_j , any commitments C_i for blinded attributes, and C_Q , along with the non-interactive Schnorr proof of the statements to be proven and any other necessary values.

The LA can verify this credential show by recomputing V as

$V' \leftarrow (x_0 + \sum x_j m_j)P + \sum x_i C_i - C_Q$, for i where m_i is blinded, and j where m_j is revealed

and using V' to verify the Schnorr proof. Note that in the honest case,

$$\begin{aligned}
V' &= (x_0 + \sum x_j m_j)P + \sum x_i C_i - C_Q \\
&= (x_0 + \sum x_j m_j)P + \sum (x_i)(m_i P + z_i A) - (Q + z_Q A) \\
&= (x_0 + \sum x_j m_j)P + \sum (x_i m_i P + x_i z_i A) - Q - z_Q A \\
&= (x_0 P + \sum x_j m_j P + \sum x_i m_i P) + \sum x_i z_i A - Q - z_Q A \\
&= Q + \sum x_i z_i A - Q - z_Q A \\
&= \sum z_i (x_i A) - z_Q A \\
&= \sum z_i X_i - z_Q A \\
&= V
\end{aligned}$$

Generally, credential shows in Lox involve some combination of hidden and revealed attributes. In particular, the ID Φ is always revealed in the existing Lox protocols, and after each protocol (or pair of protocols if two go in sequence), the credential is replaced with a new credential bearing a new ID (jointly created by the LA and the user in such a way that the LA does not learn the ID during issuance, and the user cannot bias its value). Revealing the ID only during shows but not issuance and issuing a new credential with a new ID after shows preserves unlinkability of credential shows. Once an ID is revealed, it can be added to a set of observed IDs, preventing credential reuse. A credential's bucket β is only revealed when the credential is first issued at trust level $L = 0$. All other protocols, including migration to another bucket and redeeming an invitation from a friend, blind the bucket both during the credential show and during issuance of the new credential. (In these cases, the user blinds the new β but proves in zero knowledge that the new value is correct.)

2.1.2 Using Lox

We briefly describe the ways users interact with the Lox system. We do not detail every Lox protocol here.

New users who cannot be invited by existing trusted users can obtain open invitations from the Lox invitation token distributor (likely through existing bridge distribution channels: HTTPS,

email, moat, and/or Telegram). These open invitations can then be presented to the LA in exchange for a Lox credential with $L = 0$ and a β corresponding to an open-entry bucket. (The LA also provides a single bridge line directly along with this level 0 credential.) In its current implementation, the Lox invitation token distributor sets a maximum daily number of bridges that may be distributed via open invitations (currently 100) and a maximum number of times k a given β may be distributed in an open invitation (currently $k = 10$). The distribution method for open-entry buckets is currently simple: the first k users receive open invitations for one bucket, the next k users receive open invitations for a second bucket, and so on. Lox provides various suggestions for how this may be improved in the future. Lox requires a table of all encrypted buckets to be made available to users, e.g., via a public website. Users can download this table and use their β values to decrypt their own buckets. This encrypted bridge table is updated daily to provide users with new information about their buckets, including replacements for bridges that have gone offline (not due to censorship) and a bucket reachability credential, which indicates that the bucket is believed not to have been blocked by a censor. A bucket is considered to be blocked if a majority of bridges in the bucket have been blocked.

Figure 2.1 illustrates the possible level changes in Lox. If a user's open-entry bucket has not been blocked after 30 days of the user being issued their level 0 credential, the user can exchange their credential for a level 1 credential and migrate to an invite-only bucket. This invite-only bucket contains the bridge from their open-entry bucket, along with two other bridges. If a user's credential is level 3 or greater, and their bucket has been blocked (indicated to the user by the lack of a bucket reachability credential for their bucket), they can migrate to a new bucket at the cost of losing 2 trust levels. If a user's credential is level 1 or greater, and their bucket has not been blocked after a certain number of days, they can exchange their credential, along with the bucket reachability credential for that day, for a new credential with the same bucket but a trust level that is one higher. However, if they have already migrated a certain number of times because of blocked bridges, they lose the ability to level up. Leveling up to level 2 or greater also unlocks the ability to invite friends to join *one's own* invite-only bucket at trust level 1, rather than those friends having to start with an open-entry bucket at level 0. Each time the user levels up to level 2 or greater, they obtain a finite number of invitations to give to friends. After enough time, level 4 users can "level up" back to level 4 to earn more invitations.

Every day, the LA needs to learn which bridges are blocked and which are not so that it can determine which level up and bucket migration operations should be allowed on that day. The LA also uses this information to decide when to stop distributing open-entry bridges to new users. Because these are daily operations, it is not necessary to detect censorship immediately; we simply need to know whether a bridge is reachable or not each day.

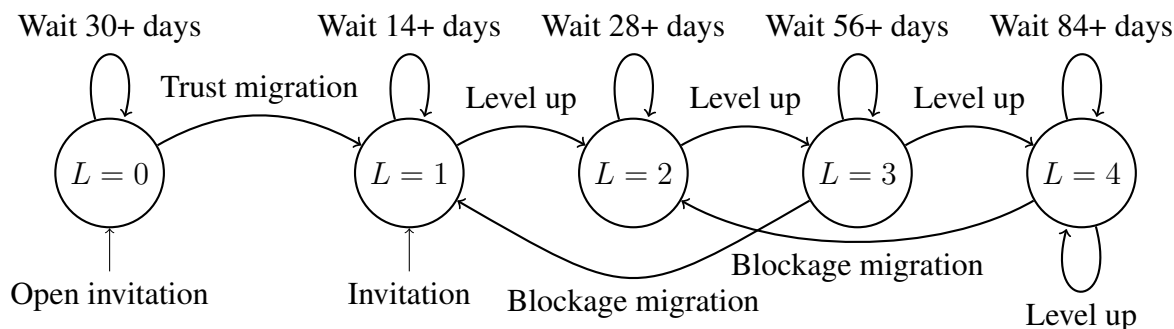


Figure 2.1: Level migrations that are possible in Lox. While it is not possible to have a level higher than 4, level 4 users can “level up” to new level 4 credentials to acquire more invitations. Leveling up can only be done after the requisite number of days have passed at the current level.

2.1.3 Other Parameters

Global Censorship Model Currently, Lox treats bridge censorship as binary: each bridge is either blocked or not blocked. If any censor learns the bridge, all Lox users with knowledge of the bridge are distrusted, regardless of whether they reside in the country where the bridge is blocked. This is in contrast with rdsys, which has a `blocked_in` field for indicating which countries block which bridges. If it is discovered that a bridge has been blocked, rdsys should be informed of which country blocks the bridge, but Lox only needs to know that *some* country blocks it.

Bootstrapping Period Open-entry buckets are less resilient against censorship. If all users begin with open-entry buckets, a determined censor may enumerate and block all the bridges, preventing the users from migrating to invite-only buckets and taking advantage of their trust networks. Lox thus recommends an initial bootstrapping period. During this period, only trusted users should be allowed to join the system, and they should be able to obtain trusted credentials.

2.2 Measuring Censorship

As established, knowledge of which bridges are accessible and which are not is a requirement for reputation-based bridge distribution systems such as Lox. A number of ideas for determining bridge reachability have been discussed [Din11a, TP22, TP23b], but the Tor Project has not currently adopted a procedure for automatically determining this. Creation of a system to make

these determinations would thus be helpful to the deployment of Lox (or similar reputation systems). We consider how our system should identify this censorship.

In a 2011 technical report [Din11a], Dingledine describes five possible sources of data for determining whether or not bridges are reachable:

1. Direct Scans
2. Reverse Scans
3. Indirect Scans
4. Bridge Statistics
5. User Reports

The first three are active approaches that would involve running some additional tests. The last two are passive approaches that do not involve introducing additional tests, instead collecting data produced as a byproduct of routine interactions between users and bridges. We discuss these proposed sources of information and relevant prior work in these areas.

2.2.1 Direct Scans

An obvious method for testing whether or not a bridge is accessible from a given region is simply attempting a connection to the bridge from that region, a method called *direct scanning*. However, performing such censorship measurements in a manner that is both safe and accurate can be challenging. If we ask volunteers in a region to perform such tests for us (i.e., intentionally attempt to access resources on a list of potentially forbidden resources), those volunteers might face retaliation. It may be possible to perform these measurements safely or mitigate these risks, for example by making the measurements appear as though they were generated by malware rather than intentional human action, as proposed by Jones and Feamster [JF15]. We find such ideas interesting but ultimately feel that asking humans to perform additional measurements for us is not reasonable.

An approach that avoids risk to human participants is using infrastructure that is not operated by end users, such as VPNs [NCW⁺20]. However, this approach carries its own concerns. The censor may apply different policies for businesses than it does for people; a resource might be accessible from a VPN run on this infrastructure but inaccessible to end users on residential Internet service plans. Furthermore, VPN providers are known for misrepresenting the true locations of their servers, and it can be challenging to properly verify these claims [WCC⁺18].

We may be unable to find commercial VPN exits that are (actually) located in the regions we wish to test. Additionally, gaining access to such infrastructure carries financial costs. Renting infrastructure such as virtual private servers (VPSes) in regions of interest carries similar benefits and limitations.

Since the particular application of censorship measurement carries an additional requirement to avoid revealing the tested resources (Tor bridges) to censors, direct scanning raises an additional concern. If we have a small number of vantage points for direct scanning from a region (such as VPN exits or VPSes we rent), and we test many bridges, we may draw attention from the censor. A vantage point for testing bridge censorship identified as such may reveal new bridges to a censor that watches its traffic [Din11b], ultimately causing more harm than good. Possible defenses against censors discovering bridges in this way include using many vantage points, limiting the number of bridges tested by each vantage point, and adding cover traffic to obscure what the vantage point is testing.

In addition to the other problems discussed, performing direct scans of many bridges (for example, all the bridges distributed by Lox) would require our system to have knowledge of all of those bridges. This is something we would like to avoid if possible, as a list of bridges is very sensitive information, and we want to minimize the harm that might be caused by compromise of our system. Due to the various limitations of direct scanning, we aim to avoid dependence on this approach. Our system does not implement direct scans, but they could be added as an optional plugin to augment the approach we take.

2.2.2 Reverse Scans

If censorship is implemented bidirectionally (meaning both that users in the censoring region cannot connect to the bridge and that the bridge cannot connect to resources in the censoring region), then we can test censorship by having the bridge attempt a connection to a resource (for example, a popular website) in the region of interest, a method called *reverse scanning*. Dingedine suggests implementing this by building a circuit to the bridge then trying to extend this circuit to the resource in the region of interest [TP11a]. Based on the way the circuit fails (which it will, assuming the resource is not a Tor node), we can infer whether or not the bridge was prevented from sending packets to the resource. The benefit of this approach is that it uses existing infrastructure and may appear less conspicuous to the censor than direct scans.

The downside of reverse scans is that they can only detect censorship, not the absence of censorship. If a reverse scan fails, we can infer that the bridge is blocked. If the reverse scan succeeds, we cannot infer that the bridge is not blocked because it might be blocked a different way. The censorship implementation may not be bidirectional, or it may be more precise, such

as blocking the IP address and port of the bridge. This approach also assumes that the same censorship policies are applied to the resource to which we attempt a connection as are applied to regular people. Rather than indiscriminately attempting reverse scans to resources in all censoring regions, we could first test for bidirectional censorship in that region by performing the same circuit extension test from an exit node [TP10]. Only if this test fails would we then attempt a reverse scan from a bridge.

We view reverse scanning as a plausible approach. Its primary limitation is that it is not a conclusive test when censorship is not bidirectional. In the case that the reverse scan succeeds, another test (such as a direct scan) would still be necessary. Additionally, as with direct scans, reverse scans would require our system to have knowledge of all the bridges being tested. As we are aiming to avoid direct scans at this time, and we wish to avoid requiring our system to know sensitive information about the bridges being tested, we do not implement reverse scans; however, a plugin for reverse scans could be added as future work. We discuss this possibility in Section 6.2.6.

2.2.3 Indirect Scans

To address the limitations of direct scanning, there has been work on collecting censorship measurements based on the censor’s response to the actions of other hosts, a method known as *indirect scanning*. Techniques for indirect scanning include sending specially crafted requests to public servers to see whether application-level censorship is occurring via DPI [VMS⁺18, RSD⁺20] and inducing a connection between two hosts and using side channels to detect whether network-level censorship prevents this connection [ant98, EPKC10]. The latter may be useful for measuring bridge censorship.

Ensafi et al. [EKAC14] describe one such approach to test whether two hosts on the Internet can connect to each other, without controlling either host. This method sends the first host (in our case, the bridge) a TCP SYN packet with its source address spoofed to that of the second host (in our case, a resource in the censoring region). If no interference occurs, the first host sends SYN-ACK to the second host, and the second host (which did not send the initial SYN) responds with RST. By measuring incrementing IPIDs on both hosts, one can learn which of these packets succeeded. The goal is to make it appear *to the censor* as though the two hosts are attempting to communicate, so as with direct scanning, this may raise ethical concerns. Augur [PEL⁺17] makes use of this technique and specifically limits its tests to devices believed to be Internet infrastructure (such as routers or middleboxes) rather than end-user devices to reduce the potential for retribution against end users.

As with reverse scans, we cannot be confident about the results of indirect scan. The censor

could block all SYN-ACK packets for which it has not seen a corresponding SYN come from its own network. The censor could allow outgoing RST packets even to forbidden hosts (perhaps specifically to prevent this measurement). As with reverse scans, perhaps we could first learn the censor’s normal behavior by testing with a Tor exit node known to be blocked in the region. However, this test is still only useful under certain conditions, and we would require another test (such as a direct scan) as a fallback. As with reverse scans, we do not implement indirect scans at this time, but a plugin for indirect scans may be a useful area of future work, which we discuss in Section 6.2.6.

2.2.4 Bridge Statistics

The Tor Project collects daily statistics about Tor nodes, including bridges, which are identified by their hashed fingerprints. These statistics are reported by the nodes and made publicly available through the CollecTor [TP24d] service. Among the files that can be downloaded from CollecTor, the most useful for our purpose is the “extra-info” descriptor [TP24f], which in the case of bridges includes a `bridge-ips` field. This field contains a list of country codes from which the bridge saw a non-relay connection during the past day and the number of unique IP addresses (rounded up to a multiple of 8) used to connect from each such country. A 2011 report [Loe11] by Loesing suggests that observing trends in these statistics for a bridge may be useful for detecting blockages and suggests three possible approaches for determining whether or not a bridge is blocked.

Absolute threshold Loesing identifies that if the number of users connecting from a certain country falls below some minimum threshold, we may consider the bridge to be blocked in that country. For example, if the number of users connecting from a country suddenly drops to 0, it might be reasonable to assume the bridge has been blocked in that country. Loesing ultimately selects this approach and sets this threshold to 32 (corresponding to 25 or more observed connections, due to rounding), pointing out that even if a country blocks a bridge entirely, the bridge may misidentify the locations of users connecting from other countries. One reason for this, Loesing identifies, is inaccuracy in the the GeoIP database used by bridges to identify connecting users’ countries. A connection may appear to come from a country where the bridge is blocked, when actually it came from a neighboring country. We identify an additional possible reason. Within one country, a resource may be accessible by some users but blocked for others, as is the case in, e.g., India where different ISPs apply different censorship policies [SGB20].

Relative threshold compared to other countries Loesing suggests that if the fraction of users connecting from a specific country out of all users connecting to that bridge falls below a certain threshold, we may consider the bridge to be blocked in that country. As Loesing observes, one issue with this approach is that it is sensitive to changes in *other* countries. A sudden influx of users from one country that just blocked vanilla Tor may result in another country's connection counts suddenly falling below this threshold, despite no actual change in usage by people in the second country.

Estimated interval based on history If the number of connecting users decreases suddenly compared to previous days, Loesing suggests that it might be possible to infer that the bridge has been blocked. However, as Loesing observes, this is challenging if the bridge is blocked soon after it becomes available to use. If the number of connecting users from a country was always low, future low connection counts will not seem unusual. It will simply appear that the bridge is not used by many people in that country.

Ultimately, Loesing adopts an absolute threshold and proposes considering both absolute and relative thresholds in future work. While Loesing's work provides a useful basis for analysis of bridge reachability based on bridge statistics, we consider analyzing these bridge stats alone insufficient. In particular, we consider an attack that can prevent detection through analysis of bridge stats. A censor may learn a bridge line and use it to block regular people from connecting to the bridge. To avoid detection, the censor can make its own connections to the bridge from multiple IP addresses within its country (choosing, of course, not to block its own connections) to artificially inflate the bridge's connection counts. In this way, the censor can avoid detection by any analysis of connection counts alone. Due to this limitation, we do not feel it is sufficient to consider only bridge stats, but we do incorporate them into our solution.

2.2.5 User Reports

Users may submit reports indicating whether or not they can connect to bridges. As discussed in Section 2.2.1, we do not wish to put users at risk by asking them to perform direct scans for us. However, these users are already attempting connections to their bridges in the course of using Tor. Without requesting that they perform any additional tests that could put them in further danger, we may be able to learn the results of these tests they choose to perform for their own purposes. Dingledine envisions users submitting the results of their connection attempts but identifies two concerns [Din11a]. First, users may be identified by the reports. (Even if the reports do not directly contain personal information about users, it may be possible to build a fingerprint for a given user based on the set of bridges for which they submit reports.) Second,

users may submit fraudulent reports. We want to be sure that we can trust the reports submitted by the user.

Currently, there is not a standard, official procedure for users to report on bridge reachability. Reports that bridges are blocked are sometimes submitted on the Tor Project forum [Odd21] or sent to members of the Tor Project [Boc21a, Gus21]. We posit that we can design reports about bridge reachability that are safe to submit without compromising users' anonymity while also limiting the ability of malicious actors to submit fraudulent reports. We incorporate such reports as a major component of our system.

2.3 Conclusion

This chapter provides background on Lox credentials, relevant information about the operations of the Lox system and its users, and considerations for designing a system to work alongside Lox. It also provides background on measuring censorship and discusses considerations for five possible approaches to detecting when Tor bridges are blocked. In the next chapter, we will describe the design of our bridge blockage detection system, incorporating the specific considerations outlined for Lox and two of the censorship detection strategies discussed in this chapter.

Chapter 3

System Design

This chapter describes the design and goals of our Troll Patrol¹ system, which collects bridge statistics and user reports and analyzes them to infer which bridges are blocked and where. Our reference Troll Patrol implementation is written in approximately 2,200 lines of Rust code and is publicly available under a free and open source license. (See Appendix A for more information.) We include code for an optional `simulation` feature used for simulation-specific tasks, as described in Chapter 4, and we use this feature in our evaluation in Chapter 5.

We formalize reports that can be programmatically submitted and processed, and we envision that these reports will be submitted by the Tor Browser. We specify two varieties of these reports: negative reports and positive reports. *Negative reports* indicate that a user was unable to connect to their bridge (possibly suggesting a censorship event), while *positive reports* indicate that a user was able to connect to Tor using their bridge.

If users choose to submit positive reports, we expect the count of valid positive reports we receive to correlate with the count of connections reported by the bridge in the bridge statistics. When possible, we pair the two with the goal of improving accuracy. Negative reports help defend against the attack described previously in which a censor can block a bridge but manipulate bridge stats to make it appear reachable. Even if we observe high numbers of positive reports and connections according to the bridge, if we also observe a high number of negative reports, we might be able to conclude that some interference is occurring.

We first describe our threat model, then the design of our negative and positive reports, and finally considerations for analyzing submitted reports and published bridge statistics to classify bridges as blocked or not blocked.

¹The Norwegian fairy tale “Three Billy Goats Gruff” features a troll who lives under a bridge and eats would-be crossers of the bridge. Our system “patrols” to find such “trolls” that prevent people from using bridges.

3.1 Threat Model

We adopt a similar threat model to that of Lox; our adversary is assumed to be a state-level censor that aims to learn about as many bridges as possible to block them (either immediately or at some future time) and to learn the identities and social graphs of users. We consider an additional adversarial motive: a censor that is unable to learn a bridge and block it itself would still like to disrupt the Lox and/or Troll Patrol systems. If the censor is able to make the Lox Authority believe that the invite-only bridges it has distributed are blocked, even when they are not, it can prevent users from gaining trust levels. If the censor is able to hide the fact that it has blocked bridges from the LA, it can prevent users from migrating to new bridges.

We focus on bridges that may be distributed by Lox and assume that censors learn about bridges to block by querying bridge distributors or by tricking honest users into sharing their bridges. While other tactics for bridge discovery exist, such as protocol-level blocking and active probing as described in Chapter 1, they are considered out of scope for Lox [Tul22, TG23], and we likewise exclude them from our threat model. Instead, we design our system for bridges that use pluggable transports that already protect against these kinds of attacks. We also consider that once a bridge has been identified by a censor, the censor may remember the bridge, even if we observe only temporary censorship of the bridge (e.g., during a political event). Thus, once a bridge is determined to be blocked, this conclusion should not be reverted later; Lox should mark the bridge as blocked and distrust the bridge’s users. Finally, we assume that the Troll Patrol system is based in a region where its own access to Tor will not be impeded; i.e., it does not need to use a bridge.

We must consider the possibility that our powerful adversary compromises any or all components of the bridge distribution and monitoring infrastructure. In particular, we examine the potential harm from compromise of rdsys, the Lox Authority, and/or Troll Patrol. Additionally, we note that the Lox Authority and Troll Patrol may be run on the same machine. Thus, compromise of one system may imply compromise of the other.

3.1.1 Rdsys

Rdsys holds a database of bridges and serves as the backend for Lox. A censor that compromises rdsys can thus learn (and block) all the bridges known by rdsys. Rdsys only provides bridges to distributors; by compromising only rdsys and no additional infrastructure, the censor would know nothing about which users know which bridges. We neither modify rdsys nor design Troll Patrol to interact directly with it, so our work does not affect the impact of compromise of rdsys.

3.1.2 Lox Authority

As discussed in prior work on Lox [Tul22, TG23], total compromise of the Lox Authority would allow the adversary to learn (and block) all the bridges known by the LA. However, Lox protects users’ social graphs and sets of known bridges even in the event that the LA is fully compromised. We introduce modifications to Lox to allow integration between Troll Patrol and Lox. We must take care when implementing these changes to ensure that they do not give an adversary greater power to censor bridges or learn users’ social graphs or the bridges they know.

3.1.3 Troll Patrol

In case the Troll Patrol system is compromised, we aim to limit the amount of sensitive information it has. In particular, we do not provide Troll Patrol with sensitive information about the bridges known to the LA, and we do not ask users to submit this information. Thus, an adversary that compromises only Troll Patrol does not learn how to block the bridges. However, some possible extensions to Troll Patrol that we discuss in Chapter 6 would be most easily implemented by providing it with a list of bridges. If future changes do result in Troll Patrol having such a list, a censor that compromises the system can learn and block these bridges.

As we ask users to submit reports about their connection attempts, we must ensure that these reports do not compromise user privacy by containing identifying information about these users. Users should identify themselves to the system with no more granularity than identifying the country from which they attempted to connect to Tor. We should also aim to ensure that user reports do not create unique fingerprints for users, e.g., based on the set of bridges they know and report. When a Lox user submits reports, they implicitly reveal to which bucket they belong, as each bridge is only available in one invite-only bucket; however, each user of that bucket should know exactly the same set of bridges, limiting the uniqueness of this fingerprint.

While we can limit Troll Patrol’s ability to block bridges and identify or link users, if the LA trusts Troll Patrol to provide accurate information about which bridges are blocked, then an adversary that compromises Troll Patrol can disrupt Lox by submitting inaccurate information to the LA. As it does not learn sensitive information about bridges and does not hold Lox credentials, Troll Patrol cannot actually submit fraudulent user reports; however, a compromised Troll Patrol system could simply submit a claim to the LA that an unblocked bridge is actually blocked or refuse to submit a claim that a blocked bridge is blocked.

Hashed fingerprint	CC	Date	Dist
Proof of bridge knowledge			
Nonce			

Figure 3.1: The structure of a negative report, where CC is the 2-character country code, and Dist is the distributor. A complete row corresponds to 32 bytes of data.

3.2 Negative Reports

A negative report is a report that a user was unable to connect to a bridge. Figure 3.1 visualizes the structure of a negative report, with each row representing 32 bytes of data. Negative reports are fairly simple. Minimally, they must identify the bridge (which they do by including the bridge’s hashed fingerprint), the country from which the user attempted to connect (which is represented as a two-character country code), and the date of the connection attempt (as a Julian day, meaning a single integer value is used for an entire day). We additionally include the bridge distributor that was used to distribute the bridge (for example, Lox) so the Troll Patrol system knows where to send the report for verification in the event that Troll Patrol is repurposed for use with non-Lox bridge distributors. Reports should be submitted soon after the connection attempt, ideally on the same date, but we choose to continue accepting them some number of days after the connection attempt. We select three as the maximum number of days a report is valid after a connection attempt. This number is somewhat arbitrary, but we note that in the case of negative reports, users may have extra difficulty establishing a connection to submit their reports.

As described thus far, negative reports contain no secret information and could be submitted for any bridge by any party. We note that if a censor learns enough information about a bridge to block the bridge, it can most effectively disrupt the use of the bridge by simply blocking it. Thus, we are not concerned with the possibility that a censor might learn a bridge and submit negative reports for it. However, a censor that does not know enough to block the bridge may be incentivized to submit a large number of negative reports so that Troll Patrol will incorrectly determine that the bridge has been blocked. If the censor submits enough negative reports for enough bridges, and the bridge distribution system tries to replace blocked bridges, this could cause unsustainable bridge churn. When Troll Patrol is used with Lox (or if Troll Patrol is used with other reputation-based bridge distribution systems), fraudulent submission of negative reports could disrupt users’ ability to accrue reputation. To prevent the censor from submitting negative reports for every bridge, we additionally require proof that the submitter should be able to connect to the bridge.

$$\text{SHA3-256 (date || nonce || bridge line or } \beta \text{)}$$

Figure 3.2: The structure of a proof of bridge knowledge. This hash is 32 bytes long.

3.2.1 Proof of Bridge Knowledge

This proof could simply be a hash of the bridge line (which contains all the information needed to use the bridge). We somewhat arbitrarily select SHA3-256 as our hash function. For bridges distributed by Lox, β from the user’s Lox credential may be used instead of the bridge line. We use a hash of the bridge line rather than the raw bridge line to prevent the Troll Patrol system (or someone who compromises it) from learning the identities of bridges. However, we consider the possibility that the censor may learn this hash without learning the bridge line (or bucket) itself and use the same hash to submit many reports.

To prevent this, we require a nonce as an additional input to the hash. The nonce is provided as a field in the report, and it makes each report unique. If Troll Patrol receives a report with a previously observed nonce, the incoming report is simply discarded without verification. This prevents a censor that does not have knowledge of the bridge line (or bucket) from artificially inflating negative report counts by submitting duplicate reports. In order to discard reports with duplicate nonces, Troll Patrol must maintain a set of observed nonces. This set would only grow over time. To prevent boundless growth, we additionally include the report’s date as an input to the hash. This allows Troll Patrol to store the nonces used for a specific date, then purge old observed nonces once it stops accepting reports for the date they were previously used.

The structure of a proof of bridge knowledge can be seen in Figure 3.2.

3.2.2 Delivery

Honest users should not submit multiple negative reports per bridge per day, and unmodified client software should not permit them to do so. We do not currently provide any cryptographic restriction to prevent duplicate negative reports (though we suggest a Lox-specific modification to provide such a restriction in Chapter 6). In case the user is unable to submit their reports before Troll Patrol performs its analysis on the day of the connection attempt, we allow the submission of reports from recent days in the past. We choose to accept results dated up to three days in the past, expecting that each report will arrive either during this period or not at all.

Ideally these reports would be delivered to an onion service (which would provide end-to-end encryption and server authentication); however, we cannot depend on this. When a user’s bridges

are blocked, they might be able to request a new open-invitation Lox credential and bridge (or switch to a different bridge type such as meek or Snowflake) and use their new bridge to connect to Tor and submit their report. However, we do not assume that bridges distributed with open-invitation Lox credentials are resilient against bridge enumeration attacks or that users will not be blocked from using other bridge types; the user’s new bridge may also be inaccessible. We do assume that the user will be able to submit the reports somehow, noting that the report is small in size and not particularly sensitive to latency. For example, Troll Patrol might offer an email address or a domain fronted API for receiving negative reports. Lox also requires a (possibly high-latency, low-bandwidth) mechanism to contact it outside of Tor, to redeem open-entry invitations and to perform blockage migration. Troll Patrol could use the same technique that Lox uses to ensure it can be reached. We leave this implementation detail out of the scope of this work and simply assume users are somehow able to submit negative reports.

Because we do not guarantee end-to-end encryption at the transport layer, we should separately encrypt negative reports. While these reports do not contain any inherently sensitive information, if a negative report being submitted via some high-latency method is observed by a censor, the censor could copy the nonce from the report and use it in its own well-formed-but-invalid negative report. (Troll Patrol cannot validate reports itself, so only storing nonces from valid reports is not a reasonable defense.) If Troll Patrol receives a censor’s fraudulent report before the user’s genuine report containing the same nonce, the user’s report will be rejected out of hand. By encrypting negative reports in transit, we prevent this attack. Similarly, nonces must be large enough to prevent the censor from submitting such invalid reports with all possible nonces.

The public key used to encrypt these reports should rotate regularly (e.g., daily) for post-compromise security. We note that this is not for forward secrecy, as reports do not contain any private information. The only semi-sensitive information in a negative report is its nonce, and this becomes useless to the censor as soon as the Troll Patrol system receives the genuine report. We do not prescribe a key distribution method, but it should be possible for clients to obtain the new key. For example, it might be distributed in the same manner as the Lox encrypted bridge table (which is assumed to be available to all Lox users). The rotating ephemeral key should be certified with a long-term signing key, and this key should be pinned within the client software. (We do not implement this key signing at this time.) We note that the encryption scheme should not authenticate the client. If an authenticated encryption scheme is chosen, the client should generate a new random key for each encrypted report to prevent linkage of those reports.

We implement this encryption as follows. Let (x, X) be Troll Patrol’s keypair (where B is the basepoint, and $X = xB$). The client:

1. generates an ephemeral X25519 keypair (y, Y) and computes $\text{ECDH}(y, X) = xyB$

2. generates a symmetric key with HKDF using SHA3-256: $k = \text{HKDF}(xyB)$
3. generates a random IV and encrypts the report: $C = \text{AES-GCM}(k, IV, \text{report})$
4. sends (Y, IV, C) to Troll Patrol

Decryption follows intuitively. (We note that it is not strictly necessary for the IV to be random because each k should be used only once. If we want to minimize processing or data transferred, the IV could simply be 0.) At the end of each day, Troll Patrol generates its own keypair for the next day and purges old keys that will never be used again. (Keys should be stored until they cannot be used to decrypt any valid reports. Because reports are still accepted for a few days after the date they represent, keys must be stored for a few days. Recall that forward secrecy is not a requirement; the deletion of old keys is simply to reclaim disk space.)

3.2.3 Processing

Negative reports are received throughout the day and possibly during the next few days. As we do not wish to provide Troll Patrol with sensitive information such as bridge lines or bucket values, it cannot verify these reports itself. After receiving a report, successfully decrypting it, and verifying that it has a unique nonce, Troll Patrol stores the report to be processed later. At the end of the day, Troll Patrol iterates over to-be-processed reports and sends a collection for each (bridge, country, date) tuple to the bridge's distributor, which should respond with the count of valid reports from that collection. Troll Patrol stores these counts to be used to evaluate bridge reachability on current and possibly future dates.

If new reports are received after the day's analysis has been performed, the next time Troll Patrol performs its analysis, it should re-evaluate its previous decision on whether to consider the bridge blocked for each day starting with the date of the earliest new valid report. Without Troll Patrol performing this re-evaluation, a censor that learns a bridge could submit many backdated negative reports for that bridge to inflate the expected number of negative reports. Then, when the censor actually blocks the bridge and users submit legitimate negative reports, the number of negative reports might be consistent with historical data for days when the bridge was not considered blocked, leading Troll Patrol to classify the bridge as reachable.

3.3 Positive Reports

A positive report is a report that a user made a successful connection to a bridge. Figure 3.3 visualizes the structure of a positive report. Like negative reports, positive reports must minimally

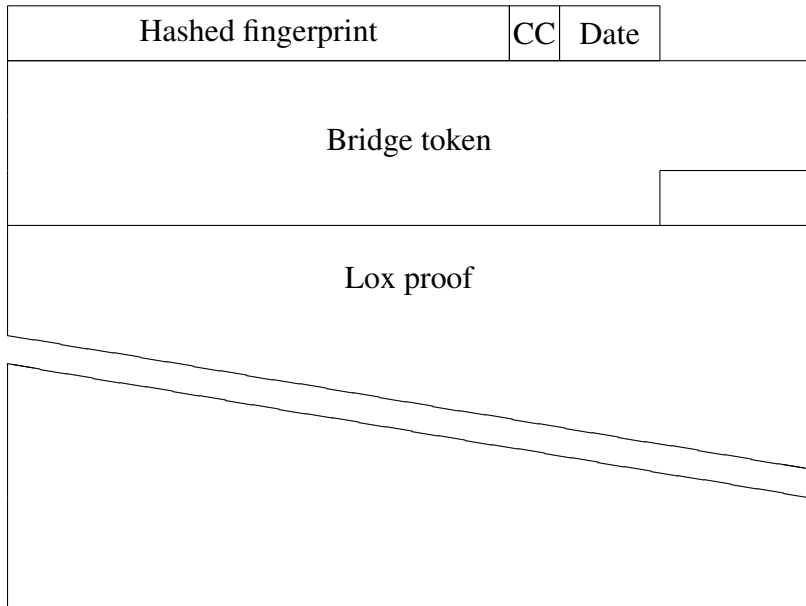


Figure 3.3: The structure of a positive report, where CC is the 2-character country code. A complete row corresponds to 32 bytes of data.

contain the bridge’s hashed fingerprint, the user’s country, and the date.

Unlike negative reports, it is a concern that a censor with knowledge of a bridge may block the bridge but submit fraudulent positive reports to prevent Troll Patrol from identifying the bridge as blocked. Thus, it is not sufficient to require knowledge of the bucket or bridge line to submit a positive report. We propose inclusion of an additional “bridge token”, distributed by the bridge, to demonstrate that a real connection to the bridge occurred. However, these tokens are both challenging to deploy and insufficient to prevent abuse, so further restriction is necessary. We additionally harness Lox’s reputation system to restrict submission of positive reports to highly trusted ($L \geq 3$) Lox users. While this restriction makes it more difficult for censors to submit positive reports, we note that it also significantly limits these reports, which can only be submitted for bridges distributed by Lox that have been active long enough to have level 3+ users.

3.3.1 Bridge Token

To prevent positive reports from being submitted when a successful connection has not been made, we propose that bridges supply a signed “bridge token” containing the bridge’s hashed

Hashed fingerprint	CC	Date
Signature		

Figure 3.4: The structure of a bridge token, where CC is the 2-character country code. A complete row corresponds to 32 bytes of data.

fingerprint, the connecting user’s country (as detected by the bridge), and the date. We consider that the user’s country may differ from the country detected by the bridge. This should not disqualify a report, as GeoIP databases are not always accurate, and we would like for it to be possible for a user to manually specify the country from which they connect, overriding the bridge token’s country field. However, the Troll Patrol system may wish to place some restrictions on this discrepancy. For example, we might accept reports if the detected country neighbors the reported country but not if the two are on different continents. As bridge tokens are not currently supplied by Tor bridges, we design Troll Patrol so that initially, positive reports can be submitted without them. If they become a standard feature of Lox-distributed bridges in the future, the operator of the Troll Patrol system will be able to simply change a boolean and require them going forward.

The structure of a bridge token can be seen in Figure 3.4. The signed data in the bridge token is all contained within the report. To reduce the size of data transmission, rather than include a full bridge token as described, users could supply only the bridge’s signature over its hashed fingerprint, the user’s detected country code, and the date. Troll Patrol could then reconstruct the token from the fields in the user’s report and verify the token’s signature. In the case that the user’s country differs from the country detected by the bridge, the user would need to specify both country codes in their report so that Troll Patrol could both verify the bridge’s signature and treat the report as coming from the user’s actual country.

By requiring a bridge token in each positive report, we can ensure that a successful connection to the bridge actually occurred. However, we observe that these tokens are insufficient to protect against a censor that wishes to hide its censorship. A censor could block regular users from connecting to a bridge but make its own connections (choosing not to block itself) to obtain bridge tokens and submit positive reports. Thus, additional protection is required.

3.3.2 Positive Report Proof

We limit positive reports to trusted Lox users, specifically users with Lox credentials with trust level 3 or greater. This requirement is enforced with a zero-knowledge proof. In order to support positive reports, we define a new Lox protocol in the main Lox library. Unlike other Lox protocols, this new protocol does not reveal the user's credential ID and does not result in the issuance of a new credential. Instead, the user simply uses their credential to issue a zero-knowledge proof of the fact *that they have a credential* that can be used to submit positive reports for the given bridge.

This protocol has no response from the server. The client simply produces a proof, and the server either succeeds or fails to verify it but sends nothing back to the client in either case. Thus, the client can include the non-interactive proof in a positive report sent to the Troll Patrol system to be later forwarded to the LA for verification.

Positive Report Proof Details

We introduce a new Lox credential show protocol to prove that the user is allowed to submit a positive report for the given bridge. Unlike existing Lox protocols, this does not require revealing the ID Φ or issuing a new credential to the user. Instead, we blind all attributes of the credential, and the LA simply verifies a Schnorr proof of the commitments to these attributes and the additional statements being proven, as well as the MAC on the credential.

The user must prove that they have a credential that should be allowed to submit a positive report for the bridge being reported. Specifically, this is a proof that:

1. The user has a valid Lox credential
2. The user's credential is level 3 or 4
3. The user's credential contains a particular bucket (which the LA must verify separately to contain the bridge being reported)

These statements together provide a stronger limitation: Not only must the user be highly trusted; they can only use their trusted credential to report bridges represented by *that credential*. This prevents a scenario where a user's bridges become blocked, the user migrates to a new bucket (causing their trust level to decrease below 3), and the user then uses their old level 3+ credential to submit positive reports for their new bridges. This proof does not directly reveal any of the fields in the Lox credential, but it allows the Lox Authority, which knows the bucket

corresponding to the reported bridge, to verify that this is the user's bucket. It also reveals that the credential's level is 3 or 4 but not which of those two values. When verifying a positive report, the LA must separately verify that the bucket used in this proof contains the bridge being reported.

We now discuss the way in which each statement is proven.

The first is accomplished by verifying the MAC. The user forms Pedersen commitments to all of their attributes m_i and to Q and computes V :

$$\begin{aligned} z_i &\stackrel{\$}{\leftarrow} (\mathbb{Z}/\ell\mathbb{Z})^* \\ C_i &\leftarrow m_i P + z_i A \\ z_Q &\stackrel{\$}{\leftarrow} (\mathbb{Z}/\ell\mathbb{Z})^* \\ C_Q &\leftarrow Q + z_Q A \\ V &\leftarrow (\sum z_i X_i) - z_Q A \end{aligned}$$

Note here that z_L is used to blind m_L : $C_L = m_L P + z_L A$. We will reuse z_L in the next step.

The second statement is proven using a range proof. The user proves that their credential's trust level m_L is 3 or 4 by selecting $g \leftarrow 4 - m_L$ and proving that $g + m_L = 4$ and $g \in \{0, 1\}$. Recall that ℓ is prime. Thus, $\forall g \in (\mathbb{Z}/\ell\mathbb{Z}), g^2 = g \iff g \in \{0, 1\}$. To prove g is a bit, the user first computes commitments to g and g^2 as described here:

$$\begin{aligned} w_g &\stackrel{\$}{\leftarrow} (\mathbb{Z}/\ell\mathbb{Z})^* \\ z_g &\leftarrow z_L \\ y_g &\leftarrow w_g + g \cdot z_g \\ C_g &\leftarrow gP + z_g A \\ C_{g^2} &\leftarrow gP + y_g A. \end{aligned}$$

Note that $z_g = z_L$, the blinding factor for m_L . In their Schnorr proof, the user proves the following:

$$\begin{aligned} C_g &= gP + z_g A && (C_g \text{ is a commitment to } g) \\ C_{g^2} &= gC_g + w_g A && (C_{g^2} \text{ is a commitment to } g^2) \\ C_{g^2} &= gP + y_g A && (C_{g^2} \text{ is a commitment to } g) \end{aligned}$$

The user submits C_{g^2} in their request. The LA recomputes C_g as $C_L - 3P$ and uses it to verify

the Schnorr proof. Note that

$$\begin{aligned}
C_L &= m_L P + z_L A \\
&= (3 + g)P + z_L A \\
&= 3P + gP + z_L A \\
&= 3P + (gP + z_L A) \\
&= 3P + C_g
\end{aligned}$$

Because C_{g^2} has been proven to be a commitment to both g and g^2 , it must be that $g = g^2$, and therefore $g \in \{0, 1\}$.

The proof for the third statement requires an additional public value, $H \in \mathbb{G}$. A new H is generated deterministically each day based on the date, and the same H is used by all users. This H must be generated with hash-to-group [BHKL13] to ensure that the discrete logarithm relationship between H and A is unknown. The user computes $D \leftarrow m_\beta H$ and submits this value. They also include the condition that $D = m_\beta H$ in their Schnorr proof. By verifying the Schnorr proof, the LA can be sure that the bucket value in the user's credential was used to produce D , but the LA must still verify that D is correct. The LA knows which bridges are in which buckets; it separately retrieves the bucket value corresponding to the bridge being reported and verifies $m_\beta H \stackrel{?}{=} D$.

Preventing Duplicate Reports

In the current design, it is possible for one user to submit multiple positive reports for one bridge on one day. It may be desirable to limit these reports to one per credential, which can be accomplished in the following way: We add a unique, never-revealed field γ to the Lox credential, and we restrict reports based on this γ . Whenever the user requests a new credential, this field should carry over to the new credential. Otherwise, each time the user leveled up or invited a friend, they would gain the ability to submit an additional report per bridge per day when they are issued a new credential through the protocol. When the user invites a friend, the invited friend's credential will have a new γ , and (after leveling up to level 3) they will be able to submit their own positive reports. We note that a censor could issue many invitations to itself and wait until it has leveled up many credentials to level 3 or higher; however, doing so would require the censor to spend more time gaining trust. If the censor is willing to wait to block a bridge until it has the ability to submit a certain number of positive reports for that bridge, this restriction would increase that delay, allowing users to connect unimpeded for longer.

In order to limit reports based on γ , we add three additional reference points, H_0 , H_1 , and H_2 , corresponding to the indices of bridges in the bucket. We add an index i , a point D_i , and

the condition that $D_i = m_\gamma H_i$ to the proof. When the LA verifies the proof, it must additionally verify that the bridge being reported has the index i in its bucket and that it has not seen D_i before. Troll Patrol could also reject reports with already-observed D_i values, as it does with duplicate nonces for negative reports.

We observe that this additional restriction does not prevent users from migrating to new buckets, then using their old credentials to submit positive reports for bridges in their old buckets. However, there is no harm in this fact. In order to migrate, the LA must have already marked the bucket as blocked, so new reports for its bridges become irrelevant.

Another observation is that a single bridge may actually be contained within two different buckets: the open-entry bucket containing only that bridge and the invite-only superset bucket containing the bridge along with two others. However, this does not mean that a user can submit two positive reports for the same bridge. In order to submit a positive report using a specific bucket value, the user must have a Lox credential with that bucket value and a trust level of 3 or 4. Values for open-entry buckets are only contained within level 0 credentials; in order to attain level 1, the user must migrate to the new invite-only bucket.

Association of Multiple Bridges in One Bucket

If, in a single day, Troll Patrol receives positive reports for multiple bridges sharing the same D value in their positive report proofs, then Troll Patrol (or an adversary that compromises the system) can infer that these bridges are in the same bucket. We consider reusing D in fraudulent positive reports for other bridges as a possible defense. Since Troll Patrol cannot verify reports itself, these could act as “cover traffic” to prevent Troll Patrol from associating bridges together by their shared use of D . However, this defense would be insufficient. Troll Patrol could simply ask the LA to verify the reports one-by-one and associate the bridges in successfully verified reports that share the same D . If preventing this association is a priority, these fraudulent cover reports could be submitted, and the LA could refuse to verify a collection of positive reports for a given bridge, country, and date more than once per day, requiring Troll Patrol to submit all of its collected reports at once. We do not consider the impact of this association significant and do not protect against it at this time.

3.3.3 Delivery and Processing

As with negative reports, positive reports are accepted if their dates are no more than some threshold number (which we set as three) days in the past, and analysis should be redone if new reports have been received and successfully verified for past days. We note that it is less

important to accept positive reports with dates more than one day in the past. If a user is able to connect to Tor, it should not be an issue to submit a positive report soon after making a successful connection. Our choice to use the same threshold for both types of reports is made for simplicity.

3.4 Analysis

Having collected its data, Troll Patrol needs some way to draw conclusions from it.

3.4.1 Possible Approaches

In Section 2.2.4, we discussed Loesing’s report on testing bridge reachability [Loe11], which analyzes only bridge statistics, not user reports. Loesing discusses three approaches: an absolute threshold, a relative threshold compared to other countries, and an estimated interval based on history. We consider how these approaches could be applied with the addition of user reports.

Absolute threshold Loesing ultimately adopts an absolute threshold due to the simplicity of this approach. One option would be to set a single threshold for negative reports, in much the same way. If we receive a requisite number of negative reports (possibly even as few as one), that may always be sufficient to conclude that a bridge is blocked. However, we note that users being unable to connect may not reflect an intentional act of censorship. A number of factors might cause a user to be unable to connect to a bridge at a particular point in time, such as networking issues on the user’s end or an organizational firewall operating on an allowlist. If a single threshold is used, it must be selected carefully to ensure that real blockages are detected as often as possible without falsely detecting random connection failures as censorship.

Rather than being a fixed number, the threshold could be relative to connection counts from bridge statistics. For example, we might consider the bridge blocked in a country if we receive at least $n/8$ negative reports for that bridge, where n is the number of connecting users according to bridge statistics. However, this could be defeated by a censor making a very large number of connections to the bridge, thus making the requisite number of negative reports to consider the bridge blocked very high. We recommend combining the two tests, considering a bridge blocked if its count of negative reports exceeds some fraction of connection counts or some absolute threshold, whichever is lower.

Relative threshold compared to other countries This approach is not very attractive for bridge connection counts due to its sensitivity to changes in other countries; however, we still consider its applicability for user reports. It may be challenging to consider report counts relative to other countries, due to different needs and cultures of people in different countries. For example, if a country is unlikely to punish its residents for circumventing censorship, users may be eager to submit reports to ensure that their circumvention tools work as effectively as possible. On the other hand, if a country has a reputation for harsh repercussions for censorship circumvention (or if there is simply a strong cultural emphasis on privacy in that country), users may be much more hesitant to submit any data about their use of the system. (We intentionally design reports to avoid identifying users; nevertheless, this telemetry should never be collected from users without their consent.) Due to these possible differences, for the purpose of evaluating bridge reachability based on report counts, we choose not to consider the number of reports received from other countries.

Estimated interval based on history This approach is attractive, as it allows us to account for differences between bridges, rather than assuming that all bridges have similar usage. However, as Loesing identifies, if a bridge is blocked very soon after being distributed, we will not have historical data on normal usage. We may conclude that a small number of connections or a large number of negative reports is standard for that bridge and fail to mark it as blocked.

We propose some combination of these techniques. A crude test such as absolute threshold should be used initially, to infer whether or not a bridge is blocked while we lack historical data. After we have collected enough data about normal usage and are reasonably confident that the bridge has not been blocked yet, we can begin using an estimated interval based on history. In the next section, we will discuss how this might apply to Lox-distributed bridges.

3.4.2 Analysis for Lox Bridges

For our purposes, Lox-distributed bridges effectively have three stages of life. It should be possible to evaluate bridge reachability with greater precision in later stages than in earlier ones.

Stage One

The first stage is the period when the bridge is being distributed to new users in an open-entry bucket. We expect that bridges are most likely to be blocked during this stage, and we will also have the least historical data on connections to them. Thus, we must primarily draw our conclusion based on generic rules, such as employing an absolute threshold, rather than comparison to

past trends. As bridges in this stage are new, we expect that they may have very low user counts. We emphasize that this does not mean that the bridge is inaccessible. We do not want to set a threshold too low such that all new bridges will be marked blocked immediately. Instead, we propose assigning greater weight to negative reports during this stage, as these explicitly indicate a failed attempt to connect.

Recall that positive reports can only be submitted by users with trust level 3 or greater. In stage one, all users of the bridge are assumed to have trust level 0, so we will not have positive reports available for our analysis. We do not necessarily want a single negative report to result in a conclusion of censorship. As discussed previously, a number of factors may prevent users from connecting to their bridges. However, just one report might be sufficient in some cases. If the bridge never receives a single connection from a country, but we observe a positive number of negative reports for that bridge from that country, it is reasonable to conclude that the bridge is blocked. We propose setting a maximum threshold for the acceptable number of negative reports and considering a bridge blocked if the number of valid negative reports we receive for that bridge exceeds this threshold. Additionally, for bridges with low connection counts, we propose scaling the negative report threshold down based on the connecting user count according to bridge stats. It should be possible to reduce this threshold in this way to allow a blocked bridge with few users to be detected even if the number of negative-report-submitting users is very small, but the threshold should never be increased above the maximum. If the threshold were to be increased based on bridge stats, then the censor could make a large number of connections to prevent detection.

Stage Two

The second stage begins after 30 days. When the bridge has remained unblocked for 30 days, the Lox Authority stops distributing the open-entry bucket and adds the bridge to an invite-only bucket. After this, we believe that the bridge is less likely to be blocked, as the censor must now learn about it from an existing user or through trust migration from an already-identified open-entry bucket. In the latter case, the censor must have been patient and avoided blocking the open-entry bridge for 30 days. By the time a bridge becomes invite-only, we also have historical data on the bridge for at least 30 days. We can use this data to provide better analysis. The minimum number of days before stage two begins should not be lower than the number of days required for the LA to stop distributing an open-entry bucket, but it may be tuned higher if this is advantageous, for example, to provide more historical data to the model used for stage two analysis. The maximum number of historical days of data to consider in the analysis is also configurable.

It is important to remember that a censor may block a bridge but make its own connections to the bridge to artificially inflate the bridge stats. To prevent this attack, it should be possible to determine that a bridge is blocked based on an increase in negative reports, even if the `bridge-ips` reported by the bridge also increase. However, we do not expect every censor to behave in this way; bridge stats can still help us detect blocking by censors that do not perform this attack. We may still be able to learn that a bridge is inaccessible based on a drop in `bridge-ips`, even if the people attempting to use that bridge do not submit negative reports. Thus, analysis in stage two should consider both signals but should allow a “blocked” conclusion based on negative reports to prevail over an “unblocked” conclusion based on bridge stats.

Stage Three

During the first two stages, we infer whether the bridge has been blocked based on bridge statistics and negative reports, but not positive reports. The third stage begins some configurable number of days after receipt of the first valid positive report (from a level 3+ Lox user). The inclusion of non-zero historical counts of positive reports is the only difference between stages two and three. These can be used to support conclusions made based on bridge stats (with the rationale that censors are less likely to become highly trusted users, so if the number of positive reports submitted by highly trusted users correlates with `bridge-ips`, then it is likely that the bridge is accessible). While evaluating based on positive reports may help protect against a censor that aims to hide the bridge’s censorship, it may still be beneficial to evaluate based on negative reports, in case a censor does acquire level 3+ Lox credentials for the bridge.

3.4.3 Example Analysis

While we do not prescribe a specific approach to analysis, we do provide an example that could be used. We use this approach when evaluating our system in Chapter 5. For simplicity, we evaluate negative reports as totally independent from positive signals (`bridge-ips` and positive reports), but as noted above, it may be beneficial to, for example, lower the threshold for considering a bridge blocked according to negative reports if the number of observed connection counts is 0.

Negative Reports

Across all stages, we apply a simple absolute threshold test for negative reports. If the count of valid negative reports exceeds some threshold, we consider the bridge blocked. We calibrate

this threshold with a variable that we call `harshness`. We support `harshness` values of 0 through 4. If we receive more than $4 - \text{harshness}$ valid negative reports for a given bridge from a given country in one day, we consider that bridge to be blocked in that country; the maximum threshold for negative reports is 4. These values are low because bridges may have low user counts, especially when they are first distributed. Additionally, we cannot reliably infer when the number of users of a bridge increases based on bridge stats, as we assume that the censor can artificially increase this number. It might be reasonable to increase this threshold over time based on the number of days the bridge has been active, but we instead opt to use a single threshold in all stages for simplicity.

Stage One

While our stage one analysis primarily relies on negative reports, we also evaluate bridge statistics in this stage using the same `harshness` parameter. For simplicity, we suppose that each user of a bridge has a valid Lox credential for that bridge, but we note that users could share their bridge lines directly with each other in practice. If each user of the bridge holds a valid Lox credential, then based on Lox's current configuration ($k = 10$), we expect no more than 10 users to know about the bridge in a given open-entry bucket, and these 10 users are not necessarily in the same country (though it would be ideal for our analysis if they were). If 0 users connect from a single country, we should see 0 connections reported for that country. If 9 or 10 users connect from a country, we should see 16 connections reported for that country. The other 8 expected values will all result in 8 connections reported for a given country. In other words, if a country has at least one active user of the bridge, we expect the reported value for that country to be 8 most of the time. If the number of reported connections falls from 8 to 0, this could reflect a single real user who simply decided not to connect on that day.

We outline a decision-making process and parameterize it with `harshness`:

- If `harshness` is 4, then we consider the bridge blocked if it reports 0 connections.
- If `harshness` is 2 or 3, then we consider the bridge blocked if it reports 0 connections and has ever reported 16 or more connections.
- If `harshness` is 0 or 1, then we never consider a bridge blocked with this test; Troll Patrol will only classify a bridge as blocked in stage one due to negative reports.

Our analysis in stage one is illustrated in Figure 3.5.

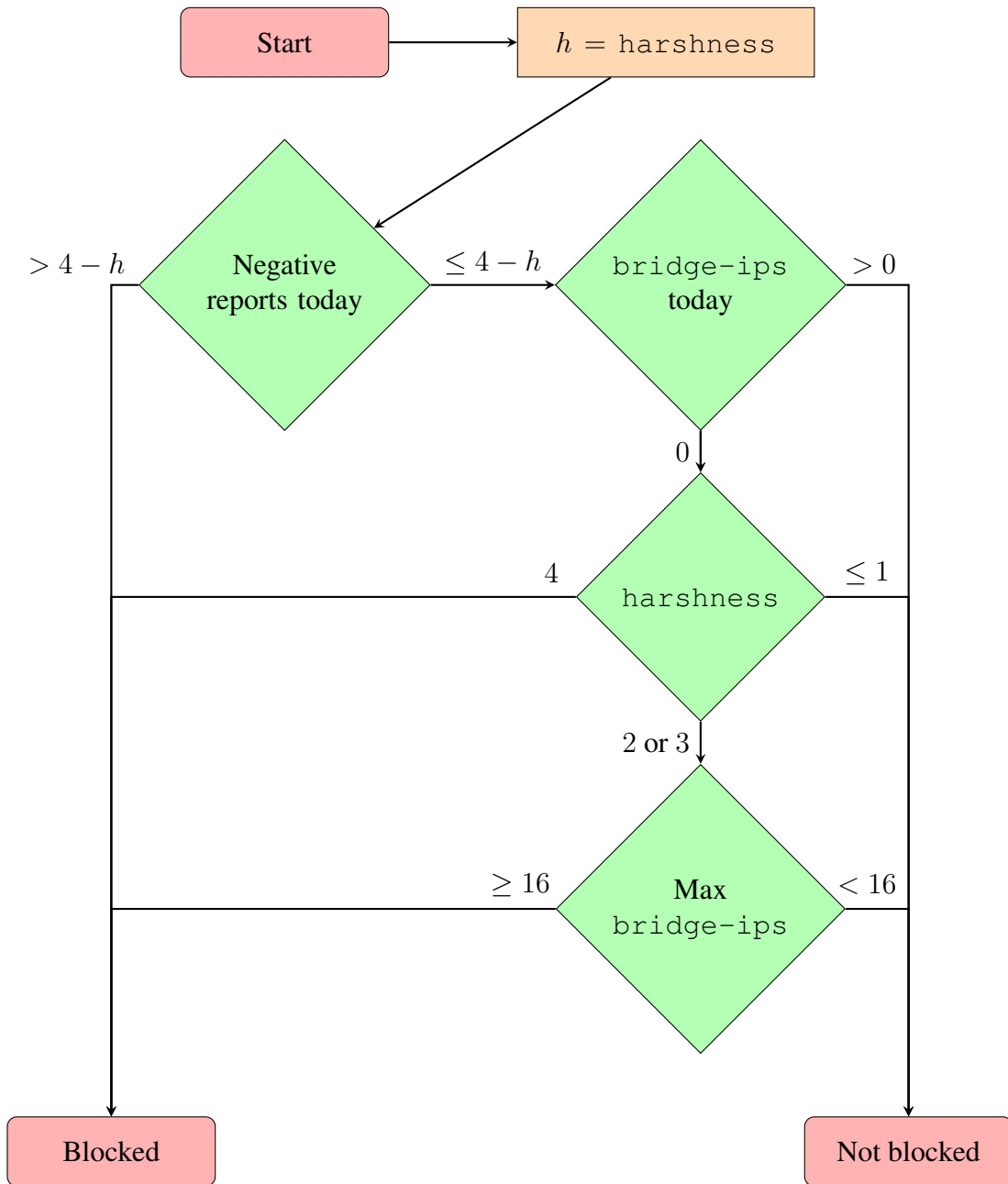


Figure 3.5: Example analysis for stage one.

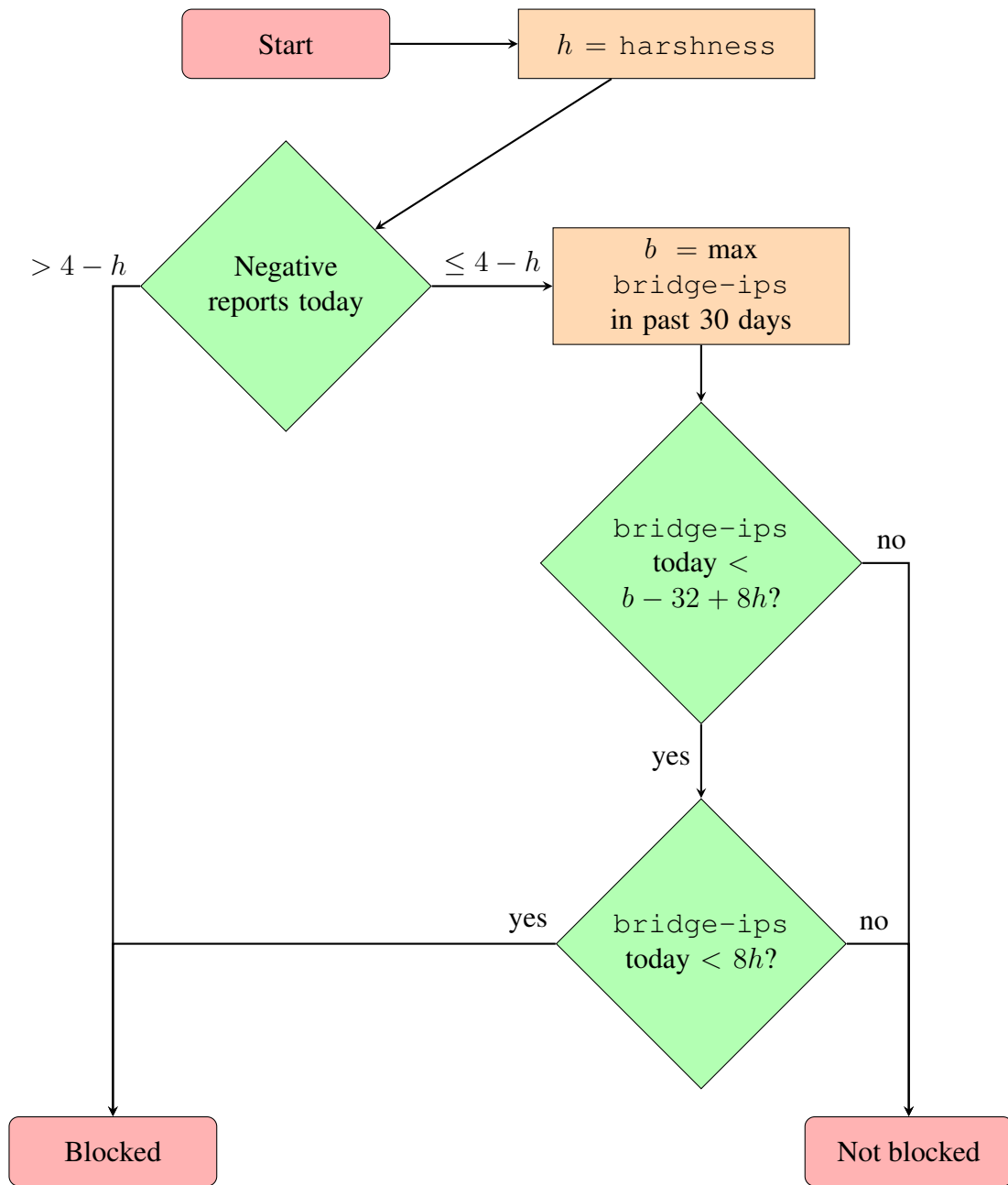


Figure 3.6: Example analysis for stage two. Stage three has the same process overall, but the thresholds for `bridge-ips` may be further reduced based on positive reports.

Stages Two and Three

In stages two and three, we define a threshold for reported connection counts based on the bridge's recent history. We consider setting our threshold to 32, the absolute threshold used by Loesing's previous study [Loe11]. However, if the bridge has always had lower connection counts, we should not consider it blocked for failing to reach this number. Thus, we take the minimum of the greatest number of connections observed over the prior 30 days and 32. We then reduce this threshold according to our `harshness` value. As connection counts are measured in multiples of 8, we reduce our threshold by $(4 - \text{harshness}) \cdot 8$. If the connection count for a given day is below this threshold, we consider the bridge blocked. We note that if `harshness` is 0, it is impossible for this analysis to detect censorship based on reported bridge statistics. If configured this way, Troll Patrol must rely entirely on negative reports. Figure 3.6 illustrates this decision-making process.

In stage three, we may have received positive reports from users on the current day; if we have, we further reduce our threshold by 8 for every 8 valid positive reports received, rounding the number of positive reports up to a multiple of 8. For example, if we receive 1 to 8 positive reports, we reduce the threshold by 8. We do not attempt to detect censorship based on a decrease in positive reports; thus, negative reports are our only defense against a censor that makes many connections to a blocked bridge from multiple IP addresses.

3.5 Conclusion

This chapter describes the threat model we assume for Troll Patrol and the design of the system. Troll Patrol collects two types of reports: negative reports, which indicate a failure to connect to a bridge, and positive reports, which indicate a successful connection. It uses these reports along with approximate connection counts reported by bridges to infer whether or not a bridge has been blocked by a censor. We discuss considerations for the analysis used to make this inference and provide a simple algorithm for the analysis. We will use this algorithm to evaluate Troll Patrol using a simulation that we describe in the next chapter.

Chapter 4

Simulation

In order to evaluate our system, we design a simulation in which Lox users join via open-entry invitations or invitations from trusted users and accrue trust over time, while censors attempt to learn about bridges and block them. In this chapter, we discuss the choice to evaluate in simulation and describe the design of our simulation.

It is more appropriate to evaluate our system using a simulation rather than experiment on the live Tor network. Performing a live test would require ample time to collect meaningful daily data for all three stages of analysis and some way to ensure that censors blocked some of the bridges during that time so we can ensure that Troll Patrol detects this blockage. It would require cooperation from users, who would either need to run a modified Tor client that submits Troll Patrol reports or manually test their bridges and submit reports. Especially given that we must somehow ensure that some bridges are discovered and blocked for this experiment to be successful, such a test may put these cooperating users at risk.

We could ignore reports altogether and evaluate Troll Patrol’s ability to identify blocked bridges based on publicly available bridge statistics alone. In this case, Troll Patrol’s classifications could be evaluated against “ground truth” collected via direct scans from a VPS vantage point or indirect scans. However, the inclusion of reports is one of the primary contributions of this work, and we hypothesize that negative reports in particular defend against an attack that Troll Patrol otherwise does not prevent. Thus, to properly evaluate Troll Patrol, it is necessary to include user reports, making a live test impractical for the reasons described above.

Our simulation involves a number of components: `rdsys`, the Lox Distributor, bridges, an extra-infos server, users, a censor, and Troll Patrol. The simulation driver is provided in the `lox-simulation` codebase and consists of approximately 1,500 lines of Rust code. It also

makes use of a Lox client library we wrote for this purpose, consisting of approximately 300 additional lines of Rust code. This driver runs alongside the Tor Project’s `rdsys` code, unmodified, as well as the Tor Project’s Lox Distributor and our Troll Patrol systems with small modifications (enabled with the `simulation` feature), most notably to allow the simulation driver to artificially increase simulated time for these two components. We provide a Dockerfile to compile all necessary components, and we use this Dockerfile to build the image used for our experiments in Chapter 5. All components run in the same Docker container, and all network calls between components are made as local connections. No part of our simulation models Tor traffic in detail; we are concerned with whether or not users are prevented from connecting to bridges, not the specific networking details of establishing or blocking these connections. We provide links to our code in Appendix A.

This chapter describes the design and role of each component and the overall operation of the simulation.

4.1 Rdsys

Rdsys provides bridges to the Lox Authority. Our simulation runs the Tor Project’s actual `rdsys` code, unmodified, with 3,600 simulated bridge descriptors, all of which are allocated to Lox. We select 3,600 bridges because this is the number of bridges used to evaluate Lox [Tul22, TG23], based on the actual number of bridges in Tor’s bridgepool at that time.

4.2 Lox Distributor

The Lox Distributor runs the Lox Authority and the Lox invitation token distributor. It retrieves bridges from `rdsys` for the LA to process. The LA partitions these bridges into buckets and performs the various Lox protocols with users. The Lox Distributor also receives collections of both negative and positive reports from Troll Patrol, verifies the reports, and returns counts of the valid reports. When Troll Patrol has determined that a bridge is blocked somewhere, it reports this to the Lox Distributor. The LA then marks the bridge as blocked and generates a migration credential for its bucket if applicable. In its current implementation, the LA does not consider *which* censor blocks the bridge; if the bridge is blocked somewhere, it considers it blocked for all users.

We use the Tor Project’s Lox Distributor code, with our modifications to allow it to verify user reports and mark bridges as blocked when Troll Patrol indicates that they are. We make

additional changes for the purpose of simulation, namely providing a mechanism for the LA and Lox invitation token distributor to artificially advance their own clocks during the simulation. We set the LA to allocate half of its bridges (1,800) as the sole bridges in open-entry buckets and half (1,800) to hot spare buckets (resulting in 600 hot spare buckets with 3 bridges each). This is the default setting for the Lox Distributor and the setting used in Lox’s evaluation [Tul22, TG23].

4.3 Bridges

Rdsys provides bridge descriptors to the Lox Distributor, which in turn parses them into bridge lines and makes these bridge lines available to users with Lox credentials. We do not rely on these bridge lines alone, however; we need to store additional information about bridges for our simulation. The simulation maintains a list of bridges that have been distributed to users with additional data about their use. These “bridges” do not represent any real implementation of a bridge. Simulated users do not perform any network calls to connect to these bridges. Instead, a bridge in the simulation is just an object with a hashed fingerprint that tracks the number of (simulated) connections made by honest users and the number of (simulated) connections made by anyone (honest user or censor) during the day. The bridge also tracks statistics for evaluating Troll Patrol:

- The date it was first distributed to any user (i.e., the date the bridge object was created within the simulation)
- The date it was first distributed to a user who is not an agent of the censor
- The date the censor first blocked the bridge
- The date Troll Patrol first detected that the bridge was blocked
- The date Troll Patrol first received a positive report for the bridge

4.4 Extra-Infos Server

One of Troll Patrol’s sources of data on bridge reachability is publicly available statistics it downloads from CollecTor [TP24d] via HTTPS. Of course, statistics for our simulated bridges are not available from the real CollecTor service, so we run a basic HTTP server locally to collect and serve this information. At the end of each day, connection data from all bridges is

formatted in a simplified version of the extra-info format (omitting most data as it is not relevant for our purpose), and aggregated to form a new record that is made available via this server. This connection data contains *only* the bridge's total number of recorded connections from each country (rounded up to a multiple of 8) including the counts of connections made by the censor in that country. If the censor makes its own connections to inflate the reported bridge connection counts, this fact is not visible to Troll Patrol when it downloads this connection data; Troll Patrol only sees the inflated number.

4.5 Censor

Censors represent countries that block access to Tor. As Lox considers a global model of censorship, where a bridge blocked in any country is treated as blocked for everyone, we perform the simulation using a single censor at a time, i.e., with all users in a single country. We consider multiple approaches to censorship and describe how each type of censor operates within the simulation.

4.5.1 Speed

While some censors will block bridges as soon as they learn about them, others may take time. This may be due to bureaucracy involved with implementing a new block, or the censor may have a specific goal it wishes to accomplish for which there is a benefit to waiting. For example, the censor might wish to acquire a highly trusted Lox credential, and it considers allowing users to access the bridge in the meantime worthwhile. Alternatively, the censor may allow access to Tor via bridges (or even publicly listed entry nodes) under normal circumstances but suddenly block Tor and all identified bridges at specific times, such as before elections. We separate censors into three categories:

Fast Fast censors block bridges immediately.

Lox Lox censors wait to block a bridge until they have access to a Lox credential with trust level 3 or greater for that bridge, so that they can either submit fraudulent positive reports to hide their censorship or migrate to a new bucket when the current one is blocked.

Random Random censors wait a random number of days. This random number (ranging from 1 to 364) is selected when the censor is initialized and represents the idea that there is some political event occurring within the year. During the event, the censor blocks all bridges known to it. The event lasts some configurable number of days, after which the censor generates a new random future date for the next political event and stops blocking bridges until that future date.

4.5.2 Secrecy

Another question is whether (and how) the censor attempts to hide its censorship. We envision that a censor might block a bridge but make its own connections and/or submit false positive reports to make it appear that the bridge is still being used.

Overt Overt censors make no attempt to hide which bridges they block.

Hiding Hiding censors attempt to closely replicate normal connection and positive report numbers to avoid notice (even by manual review). Whenever a hiding censor blocks a connection to a bridge, it makes its own connection to the bridge; we simulate this by incrementing the number of total connections the bridge observes at that time but not the number of real connections from users. If the censor is able to submit positive reports for the bridge (i.e., the censor has a Lox credential with level 3 or greater with the bucket value of the bridge's bucket), it also submits these reports to hide the censorship. (The censor's connections to the bridge occur at the time it blocks the connection attempt, but the positive reports are submitted at the end of the day.) We note that this implementation is simple but has the flaw that if users naturally choose not to attempt connections to the bridge because their past attempts have not been successful, the censor will also stop making its own connections. Thus, this may not be an effective strategy for censor secrecy in our simulation. A more sophisticated censor could build a statistical model of standard user connection behavior to replicate after blocking a bridge, but we do not implement such a technique at this time.

Flooding Rather than simply replicating the number of real connections that would be made normally, flooding censors make a (configurable) large number of connections to all bridges known by them, as well as submitting a (configurable) large number of positive reports, or as many as possible. In Section 3.3, we discussed a technique for limiting positive reports to one per bridge per Lox credential per day. We do not implement this technique in our reference Troll Patrol implementation, but for the sake of simulation, we include a configuration option to enable

this restriction. In this case, the censor will submit no more positive reports for each bridge than it has Lox credentials containing that bridge’s bucket value.

4.5.3 Totality

Censors may not simply block a bridge for everyone. A censor may block a bridge for only some users, or it may apply a censorship policy that does not completely prevent access but acts as a significant deterrent (for example, by slowing down traffic to a bridge). We classify censors into three categories:

Full Full censors simply block known bridges for all users.

Partial Partial censors block only some percentage of connection attempts to known bridges. Censorship may not be uniform across a country. The censor may decide to probabilistically block only some connections, or it may be that different ISPs within a country have different censorship policies, as is the case in India [SGB20]. In our implementation, we model different ISPs implementing different censorship policies, meaning we expect that an individual user will either be in the area of censorship and thus unable to connect or not in the area of censorship and thus able to connect. (For simplicity, we do not account for users moving from place to place where they may use different networks.) The percentage of users whose connections are blocked by the censor in the simulation is configurable.

Throttling Throttling censors make connecting painful for users by slowing them down. This technique has been employed, for example, in Russia to discourage people from connecting to Twitter [XRS⁺21]. In this case, users’ connection attempts may succeed, so the bridge should report the connections in its `bridge-ips`, but some percentage of users will consider the bridge not to work properly and (if those users submit reports) submit negative reports for the bridge. We make this percentage configurable as well.

4.5.4 Bootstrapping Period

Lox recommends a bootstrapping period, during which open invitations are not publicly available, and only trusted people are invited to participate in Lox. The goal of this period is to ensure that if Lox’s open-entry mechanism is overwhelmed by the censor, users can still join invite-only bridges. Indeed, we observed in early iterations of this simulation that the censor would

quickly enumerate all the bridges available via open invitations. In fact, because open invitations are distributed to $k = 10$ users in a row, the censor’s best technique for enumerating the open-entry bridges is requesting all the open invitations that are available each day, to rotate through and learn all the bridges available that day. This results in users being unable to access bridges not because the censor blocks them but because the Lox invitation token distributor runs out of invitation tokens to distribute to prospective honest users.

We build into our simulation the assumption that the censor simply wins the open invitation game by claiming all the remaining open invitations, and we combat this with a bootstrapping period. For a configurable number of days, the censor does not act at all within the simulation. During this time, users can join normally with open invitations, use bridges, level up, invite friends, and so on. Leveling up in Lox requires long wait times; starting from level 0, a user must wait a minimum of 128 days to rise to level 4. Thus, our simulation requires a long bootstrapping period in order to build up some base of very highly trusted users. Lox could offer faster approaches to bootstrapping, such as providing trusted users with special open invitations that allow them to join with elevated trust levels, as Tulloch and Goldberg recommend [TG23]. However, such a change should be implemented in Lox, not in our simulation. After the bootstrapping period ends, the censor obtains as many open invitations as possible for one bridge (as it might learn one bridge in use by an actual user this way), and then open invitations are simply disabled for users and the censor alike for the remainder of the simulation. We discuss this decision further in Section 4.9.

One implication of open invitations being unavailable to users (either because the censor claims them all or because we disable open invitations) is that after the bootstrapping period ends and all users of open-entry buckets have migrated to invite-only buckets, the number of unblocked buckets decreases over time as the censor learns new buckets and blocks them. Without open invitations, users have no way to learn new bridges except through invitation to existing buckets or migration to new buckets. In the latter case, the previous bucket must be blocked, and all eligible users migrate to the same new bucket, causing no net increase in the number of unblocked buckets. Eventually, users are unable to migrate, either because they have observed too many blockages to level up to level 3 or because the censor blocks their buckets before they can reach level 3. These users are cut off and can only rejoin Lox via an invitation to an existing bucket. In our implementation, the censor has some positive probability of being invited to a given bucket, so over many days it learns most or all of the buckets in use. To address this, it may be necessary to provide some additional method of distributing new buckets, such as having highly trusted individuals outside the country share invitations with trusted contacts within the country. We do not implement this method at this time; such a method should be implemented by Lox itself rather than just being part of our simulation.

4.6 Users

Users have Lox credentials, which they use to learn about bridges and connect to them. Each user must have a primary Lox credential at all times, and (in the event that their primary credential's bridges are not reachable) they may have a secondary credential as well. Some users cooperate with the censor by providing bridges to block and Lox credentials to abuse. We call these users “agents” of the censor. For our simulation, we assume that the behavior and motivations of a censor agent will be very different from those of an honest user.

4.6.1 Censor Agents

We assume that the objective of a censor agent is to learn useful information (bridge lines and high-level Lox credentials) for the censor, not to actually use Tor. Whenever such a user learns a bridge line or Lox credential that might be used to submit positive reports, they provide it to the censor.

Each day, each censor agent downloads their bucket and ensures that the censor knows all the bridges in the bucket. If the agent has a Lox credential that is eligible for trust promotion or level up, they perform this protocol. If the LA has determined that their bucket is blocked, however, they attempt to migrate to a new bucket and (if successful) immediately provide their new bucket's bridges to the censor. Finally, if the agent's credential has any invitations available, they immediately use all of them to invite additional censor agents. Recall that Lox adds invited users to the inviting user's bucket, preventing the censor from learning additional bridges in this way. While this does not allow the censor to learn additional bridges, it may allow it to eventually hold more high-level Lox credentials. In practice, this may not actually be useful to the censor. We discuss this further in Section [4.9](#).

4.6.2 Honest Users

Each day, a given honest (non-censor-agent) user may or may not attempt to use Tor via bridges. The probability of a user attempting a connection is configurable. If the user decides to connect to Tor, they attempt to “connect” to each bridge in their bucket to see which ones are accessible. This choice for users to test all their bridges when connecting to Tor is consistent with current Tor client behavior. There is an open issue [\[TP24g\]](#) to limit the number of such test connections users perform; if such a change is adopted, future evaluations should be adapted accordingly.

If the censor knows the bridge, and its configuration indicates that it should block the bridge at the current time, the connection is unsuccessful. Additionally, with some configurable probability the connection will be unsuccessful even if the censor does not block the bridge. (This models, e.g., network errors.) In some rare cases, it may be beneficial to set this probability very high. A study by Zhu et al. observed significant slowdowns and packet loss for transnational Internet traffic entering China during some period of the day [ZMW+20]. This study found average daily packet loss rates from 5–15% and peak loss rates as extreme as 50%. If we were modeling China only during such periods, it may be reasonable to assume many response packets from the bridge to the user will be dropped, resulting in a high rate of connection failure. However, this is extremely unusual; in most cases, the probability should be very low. During no-slowdown time, the aforementioned study observed a 0.28% packet loss rate in connections from San Francisco, California to each of Singapore and Beijing. In the event of connection failure, users in our simulation also perform some configurable number of retries, only falsely detecting that the bridge is blocked if all attempts fail. We do not model connections in greater depth than determining whether or not they are successful; the networking details of establishing real Tor connections are irrelevant to this evaluation.

The user may submit positive reports in the event of successful connections (if the user has a level 3 or higher Lox credential) and negative reports in the event of unsuccessful connections. Rather than being a probabilistic event each time the user attempts a connection, whether or not a user submits reports is determined only once, when the user is instantiated within the simulation. If a user submits reports, that user submit reports every day they attempt to connect to bridges. This emulates users being prompted for consent when they first use Lox so that their client can automatically submit reports for them, rather than users manually submitting their reports each day. For simplicity, our simulation assumes that users consent to either both or neither type of reports, but we note that in a real deployment, users should be given the option to consent to either individually. The probability of users granting consent is configurable.

Each day the user decides to use bridges, they first re-download the bucket corresponding to their primary credential to learn any replacement bridges and determine whether they still have a bucket reachability credential. The user’s bucket is contained in the encrypted bridge table, a list of encrypted buckets that Lox assumes will be made available to all users through some mechanism that the censor does not effectively block. Thus, even if the user’s bridges are all blocked, we assume that the user can obtain their updated bucket and check for a bucket reachability credential. (For the purpose of implementation in the simulation, the user downloads this table directly from the Lox Distributor.)

The user attempts to connect to each bridge in their bucket. If these connections all fail, the user attempts to perform a blockage migration. If the user is unable to migrate, they solicit an invitation from other existing users to rejoin Lox with a new level 1 credential. For simplicity

of implementation, the simulation maintains a global pool of invitations for honest users. When users are ready to invite friends, they add an invitation to this global pool so that it can be used by the next honest user who needs one. When users attempt to redeem invitations from the pool, they try as many invitations as needed until they are able to redeem one or the pool is depleted. (If invitations are more plentiful than users requiring them, they may expire. This is why the user may need to try multiple to find an unexpired invitation.) If the user is able to redeem an invitation, they replace their old credential with the new one they obtain in this way.

If the user is unable to redeem an invitation or if they still cannot connect with their new bridges, they fall back to a secondary credential. The secondary credential is a level 0 credential the user obtains as a backup for their primary credential. As long as their secondary credential remains level 0, they do not replace their primary credential with it, instead holding both simultaneously. If the user does not already have a secondary credential, they request a new open-entry invitation token and redeem it to obtain a new level 0 credential. (Of course, they are only able to do this if open invitations have not been disabled.) The user obtains the secondary credential's bucket and attempts to connect to its bridge. If this connection also fails, the user discards the secondary credential in preparation for the next day, when they will request another.

After attempting to connect to their bridges, level up, and/or migrate, the user may invite friends. For each invitation their credential is capable of producing, the user has some configurable chance of choosing to generate the invitation. With some additional configurable probability, the user is tricked into giving their generated invitation to a censor agent, in which case this new censor agent user is added immediately to the simulation. This chance is greater for lower-level users. We apply a multiplier to the configurable probability of inviting a censor agent in this way:

- A level 2 user has a $1 \times$ probability of inviting a censor agent.
- A level 3 user has a $0.5 \times$ probability of inviting a censor agent.
- A level 4 user has a $0.25 \times$ probability of inviting a censor agent.

This is based on the assumption that honest users with higher trust levels will be more cautious with their invitations and more likely to have trustworthy friends. If the invitation does not go to a censor agent, it is added to the simulation's global pool of available invitations.

At the end of the day, some number of new users attempt to join Lox. This number is randomly generated within a configurable range. These new users first attempt to join using invitations from the global pool. If no working invitations are available, they instead attempt to join via open invitations, provided open invitations have not been disabled.

4.7 Troll Patrol

Troll Patrol receives negative and positive reports from users and downloads the aggregated extra-infos file from the past day (which contains each bridge's individual extra-info record for that day). It sends the reports to the Lox Distributor for verification and stores the resulting counts. Troll Patrol then goes through the list of bridges for which it has data and for each bridge evaluates whether or not the censor blocks that bridge. It reports the blockages to the Lox Distributor so the LA can mark these bridges as blocked. We use our reference Troll Patrol implementation in the simulation, with some simulation-specific customizations that most notably allow us to simulate the passage of time.

4.8 Simulation Operation

The simulation takes place over a configurable number of artificial days. Each day, the various parties take their actions. Users act first. Only one user acts at a time, and the order of user turns is randomized each day. Each honest user may attempt to use Tor via bridges, level up their credential, request a new credential, submit reports to Troll Patrol, and/or produce new invitations (which are either used immediately to add new censor agents to the simulation or added to the global invitation pool), as described in Section 4.6. Censor agents do not attempt to use bridges or submit reports to Troll Patrol, but they may perform Lox credential operations including inviting censor agent friends.

At the end of the day, some number of new users attempt to join the simulation by obtaining Lox credentials. They join using invitations from the global invitation pool if they are able; if not, they attempt to join with open invitations. Prospective users who are unable to obtain working invitations, for example because the global invitation pool has been depleted and the bootstrapping period has ended, are added to a backlog to try again the next day.

After all user actions have been taken, the censor performs its end-of-day tasks. If the censor attempts to hide its censorship by flooding connections, it does so at this time. (Hiding censors instead make their false connections when they block users' connection attempts.) Both flooding and hiding censors submit as many positive reports as possible, up to a configurable maximum, at this time. Then, if the censor's speed is random and its period of censorship has come to a close, it sets a new future date to begin blocking again. Once the censor has been given the opportunity to inflate bridges' connection counts, the bridges send their reports of connections for the day to the extra-infos server and reset these counts for the next day. Troll Patrol then downloads the aggregated extra-infos file containing all the bridges' connection counts, sends

the reports it has collected during the day to the Lox Distributor for verification, classifies which bridges it believes to be blocked, and sends this information about bridge blockages to the Lox Distributor. The LA marks those bridges as blocked. Finally, time is artificially advanced to the next day, and the daily tasks begin again.

4.9 Optimizations

As the number of users within the simulation grows, the simulation becomes more and more unwieldy, dramatically slowing down. In order to achieve faster simulation runs, we make two main optimizations. In this section we document these optimizations and discuss their impact on the censor's abilities.

4.9.1 Disabling Open Invitations

First, we discuss our choice to disable open invitations after the bootstrapping period. We consider that there are three ways we could reasonably model open invitations after the censor begins acting:

1. Allow the censor to request some but not all open invitations
2. Allow the censor to request all the open invitations
3. Disable open invitations

The second assumes a stronger censor than the first. Indeed, we assume in our work that the censor has great technical power and can overwhelm open bridge distribution systems. Thus, we would opt for the second option over the first. In the case that the censor requests all open invitations, it learns at most one new bridge that is distributed to honest users. After that point, by requesting new open invitations, the censor learns only new bridges that will never be distributed to honest users in open-entry buckets. However, we consider that the censor may learn new bridges by keeping its level 0 Lox credentials until it can perform a trust migration to a superset bucket that is shared with honest users. We acknowledge that by closing off open invitations, we are weakening the censor by preventing this method of infiltrating buckets.

Nevertheless, we believe that our choice is reasonable. We note that if Lox is deployed in a country where the censor is expected to enumerate open bridges, it may be prudent not to offer open invitations at all, and instead to start with bootstrapping and then just rely on trust networks,

or to disable open invitations at the first sign of censorship. We model something closer to this scenario by closing off open invitations when the censor begins acting instead of allowing the censor to claim all the remaining open invitations. We believe that our simulation is still useful for modeling post-bootstrapping Lox in an invitation-based context.

4.9.2 Reducing the Number of Censor Agents

Because one user can invite multiple friends to join their bucket, the number of users of a given Lox bucket can grow exponentially. In our simulation, this does not happen for honest users; the maximum number of prospective non-agent users grows linearly, and invitations from the global pool are only redeemed to add new users when there are prospective honest users who would like to join. However, the number of censor agents does grow exponentially if the censor is not prevented from inviting friends. The censor can learn more Lox credentials by introducing as many censor agents as possible into the simulation. However, learning new Lox credentials is only actually beneficial to the censor up to a certain point.

Censors can use Lox credentials for three purposes:

1. to migrate from blocked buckets to new buckets, thereby learning more bridges to block
2. to submit positive reports
3. to invite more censor agents and obtain more Lox credentials

The first purpose is the only one that actually benefits the censor in our simulation. In some situations, submitting positive reports may be useful to the censor, but in the sample analysis we use in our experiments (described in Section 3.4.3), a censor that attempts to hide its censorship can do so perfectly without the use of positive reports, rendering positive reports useless for the censor. Against a different classifier, it may be beneficial to submit more positive reports, and we allow the censor to hold a configurable maximum number of Lox credentials to facilitate this. The censor can invite more censor agents to obtain more Lox credentials, but doing so does not help the censor learn more bridges. Due to Lox's inheritance properties, the new agent will have the same bucket of bridges and the same number of observed blockages as the user who invited them, along with a strictly lower trust level.

With this insight, we can drastically reduce the number of censor agents the simulation needs to track. We make the following optimizations. First, we only create a new censor agent if the new agent's Lox credential would correspond to a bucket that has been distributed to honest users, has not already been marked as blocked by the LA, and for which the censor does not already

have its maximum number of useful Lox credentials for submitting positive reports. Second, if a censor agent's bucket has been marked blocked, but the censor agent is below level 3 and cannot migrate to a new bucket, we remove the agent from the simulation. Once the agent's bucket has been blocked, they will forever be unable to level up, and they lose their utility to the censor.

4.10 Conclusion

In this chapter, we describe a simulation we design. In this simulation, Lox users in a country attempt to use Tor via bridges, and a single censor attempts to learn these bridges to block the users' access to Tor. Users submit reports to Troll Patrol indicating the success or failure of their connection attempts, and bridges publish usage statistics. Troll Patrol makes daily classifications regarding bridge reachability based on these sources of data and reports them to the Lox Distributor. In the next chapter, we will use this simulation to evaluate Troll Patrol's effectiveness as a classifier of bridge reachability.

Chapter 5

Evaluation

In this chapter, we discuss our use of the simulation described in the previous chapter to evaluate Troll Patrol’s ability to detect blocked bridges. We aim to learn how reliably our classifier is able to detect censorship while avoiding misdetections and what impact the addition of reports from users (rather than only relying on usage statistics reported by bridges) has on its ability to perform this detection.

5.1 Experiment Setup

To evaluate the performance of Troll Patrol, we performed two sets of simulations. In the first, we varied the probability of users submitting reports to allow us to evaluate the impact of these reports. In the second, we varied the harshness of our classifier to evaluate its overall ability to detect censorship. We used a total of 41 unique configurations, as described below. Two configurations appeared in both experiments (resulting in 43 configurations), and for each configuration, we ran the simulation five times, resulting in 215 total trials. We divided these into 7 batches. The greatest number of trials in a single batch was 34, and the smallest number was 25. Each batch ran on an Intel Xeon E7-8860 at 2.20GHz with 18 cores (36 hyperthreads), with each trial running in its own Ubuntu 24.04 Docker container. Each trial ran for 500 simulated days. Our highest observed RAM usage (measured by Resident Set Size) during any single simulation run was around 688 MB.

In each trial, rdsys provided 3,600 bridges to the Lox Distributor. The Lox Distributor was run in its default configuration, including partitioning half the available bridges into open-entry buckets and the other half into hot spare buckets and distributing each open-entry invitation to

at most 10 users. Each day in each trial, 0 to 20 new prospective users attempted to join Lox via either an invitation from an existing user or an open-entry invitation. Each trial began with a 180-day bootstrapping period. After the censor’s first day, users stopped requesting open-entry invitations; after 180 days, the only way for new users to join Lox was through an invitation from an existing user.

We used two types of censors in this experiment: overt (making no effort to hide censorship) and flooding (making many connections to blocked bridges to hide censorship). With its current analysis, Troll Patrol detects that a bridge is blocked only if it receives a certain number of negative reports for that bridge or if the bridge’s reported connection count falls below a certain threshold. This threshold for connection counts may be reduced if Troll Patrol receives valid positive reports, but it is never greater than 32. Reported connection counts are rounded up to multiples of 8, so as long as the censor makes at least 25 connections to each bridge it blocks each day, Troll Patrol can only detect censorship if users submit negative reports. We assumed that users can always submit negative reports, so the censor can only increase connection counts (and possibly submit positive reports), making flooding its strongest possible defense against detection.

Both censors were fast (blocking bridges immediately upon learning them) and full (blocking bridges for all users, rather than blocking only a fraction of users or throttling connections instead of blocking them outright). We note that Lox limits the harm that can be caused by a patient censor. A patient censor can submit positive reports, but as discussed, these are not helpful in evading detection. A censor agent can invite friends, but these friends join the same bucket, so the censor cannot learn new bridges this way. The only real benefit of patience for the censor is that if it can obtain a credential of level 3, it can block the bucket’s bridges, migrate to the replacement bucket, and block that bucket’s bridges as well. This is an effective tactic for eventually cutting off access to Tor for the users of that bucket, but it requires the censor to first allow those users to connect unimpeded for at least 42 days while the censor levels up from level 1 to level 3. By choosing a fast and full censor, we model a censor that aims to minimize the overall number of user-hours of bridges, rather than comprehensively block every possible bridge.

In our first experiment, we varied the probability of users submitting reports and measured the percentage of blocked bridges that were successfully detected by Troll Patrol and the number of days it took to detect the blockage. For this set of experiments, we set our harshness value to 2, meaning that Troll Patrol detected that a bridge was blocked only if it received 3 or more valid negative reports for that bridge in one day or if the bridge’s daily reported connection count fell below a certain threshold that was never greater than 16. In our second experiment, we selected 0.25 as the probability of users submitting reports and varied the harshness of our classifier to gain insight into Troll Patrol’s tradeoff between precision and recall and to provide a recommendation for calibrating the classifier. Table 5.1 summarizes the configurations for both

Table 5.1: The configurations used in our experiments. We did not perform trials with a flooding censor and 0 probability of users submitting reports because Troll Patrol is incapable of detecting censorship under these conditions. Both experiments use identical configurations for both flooding and overt censors when harshness is 2 and the probability of users submitting reports is 0.25.

Experiment	Censor	Harshness	Probability users submit reports
1	Overt	2	0, 0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.6, 0.7, 0.8, 0.9, 1
1	Flooding	2	0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.6, 0.7, 0.8, 0.9, 1
2	Overt	0, 1, 2, 3, 4	0.25
2	Flooding	0, 1, 2, 3, 4	0.25

experiments.

From the design of our classifier, we know that positive reports cannot help Troll Patrol detect censorship, but we conjectured that they might allow it to avoid misclassifying accessible bridges as blocked. To test this, when Troll Patrol evaluated a bridge in stage three and concluded that it was not blocked, we had it also evaluate the bridge as if it was stage two (i.e., ignoring positive reports) and indicate if the bridge would have been classified as blocked without the positive reports.

5.2 Results

We focus on reporting statistics relevant to Troll Patrol’s ability to detect censorship rather than the efficiency of our Troll Patrol implementation, noting that Troll Patrol only needs to classify each bridge once per day, so efficiency is not a major concern. Assuming it is an accurate classifier, Troll Patrol makes many classifications on unblocked bridges but does not make many classifications on bridges after they become blocked. Specifically, once Troll Patrol has determined that a bridge is blocked, it assumes that the censor knows the identity of the bridge and will not forget this information; thus this classification should not be reverted. If Troll Patrol fails to detect that a bridge is blocked one day, it tries again the next day, but if it falsely detects that a bridge is blocked, it never revisits this decision. For this reason, we do not consider each *instance* of Troll Patrol classifying a bridge. Instead, we consider whether or not Troll Patrol makes the correct classification about each *bridge*. We consider Troll Patrol’s classification for a bridge as:

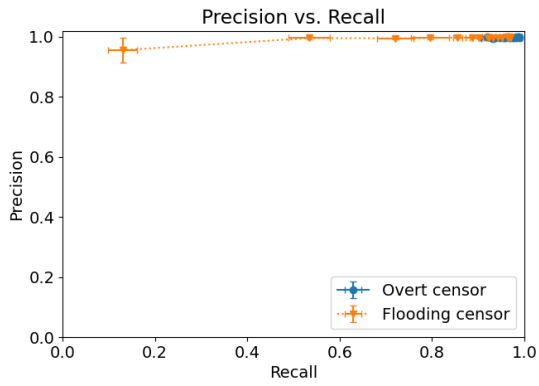
- a **true positive** if the censor blocks the bridge and Troll Patrol classifies the bridge as blocked within some maximum number of days
- a **true negative** if the censor never blocks the bridge and Troll Patrol never classifies the bridge as blocked
- a **false positive** if Troll Patrol classifies the bridge as blocked but the censor has not actually blocked the bridge (even if the censor later does block it)
- a **false negative** if the censor blocks the bridge but Troll Patrol fails to classify the bridge as blocked within some maximum number of days

If Troll Patrol correctly detects that a bridge is blocked, but it takes many days to do so, this hardly benefits users. For this reason, we set a maximum number of days for Troll Patrol’s detection to be considered a true positive, and we also present the number of days (within that range) Troll Patrol took to detect blockages. We somewhat arbitrarily select the maximum number of days for Troll Patrol to detect censorship as 10. Plots of the results from our experiments are provided in Figures 5.1 and 5.2, and the full results are provided in Tables 5.2, 5.3, 5.4, and 5.5. We plot precision (how often Troll Patrol is correct when it claims that a bridge is blocked) and recall (the percentage of blocked bridges that Troll Patrol is able to correctly identify) for both experiments. For the first experiment, we find that precision varies only within a small range, so we also plot recall on its own. We present the number of days required to detect censorship as violin plots to demonstrate the density of our results, as they are not normally distributed. From Figures 5.1c and 5.1d we can see that even if 100% of users submit negative reports, Troll Patrol is less likely to detect censorship by a flooding censor than by an overt one, and by comparing the violin plots in Figures 5.1e and 5.1f, we can see that when it does detect this censorship, it often takes longer to do so.

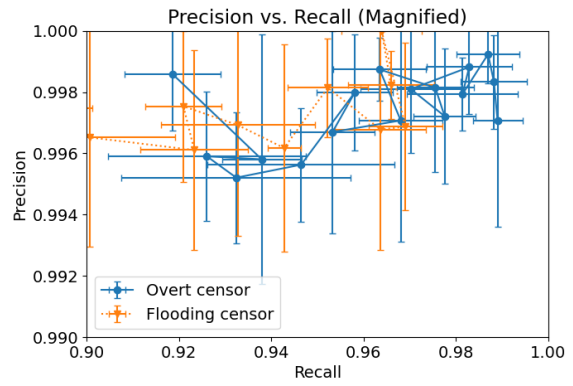
In our experiments, we observed 0 instances in which Troll Patrol classified a bridge as not blocked only due to positive reports. In other words, users submitting both negative and positive reports was equivalent in our experiment to users submitting only negative reports; positive reports had no effect whatsoever.

5.3 Discussion

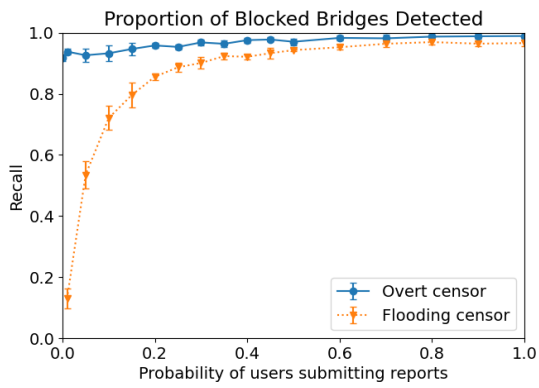
We first discuss the overt censor in the first experiment. Of the bridges that were correctly detected by Troll Patrol as blocked by the overt censor, the vast majority were detected blocked within one day, as can be observed in Figure 5.1e. We see two areas with observable density



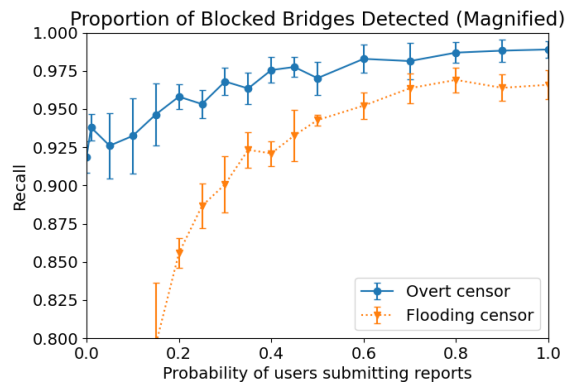
(a)



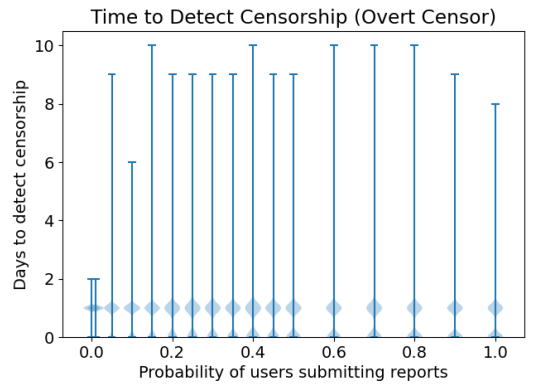
(b)



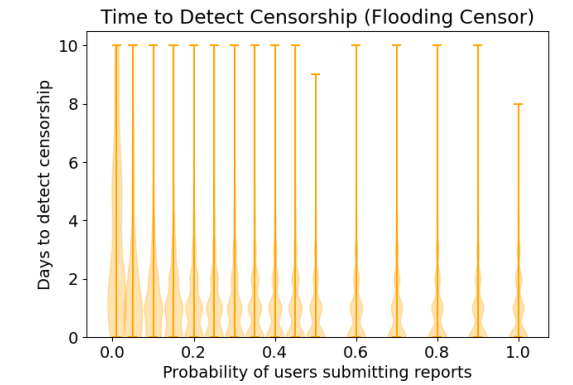
(c)



(d)



(e)



(f)

Figure 5.1: These plots show data from our first experiment. Troll Patrol’s harshness is held constant at 2, and the probability of users submitting reports is varied.

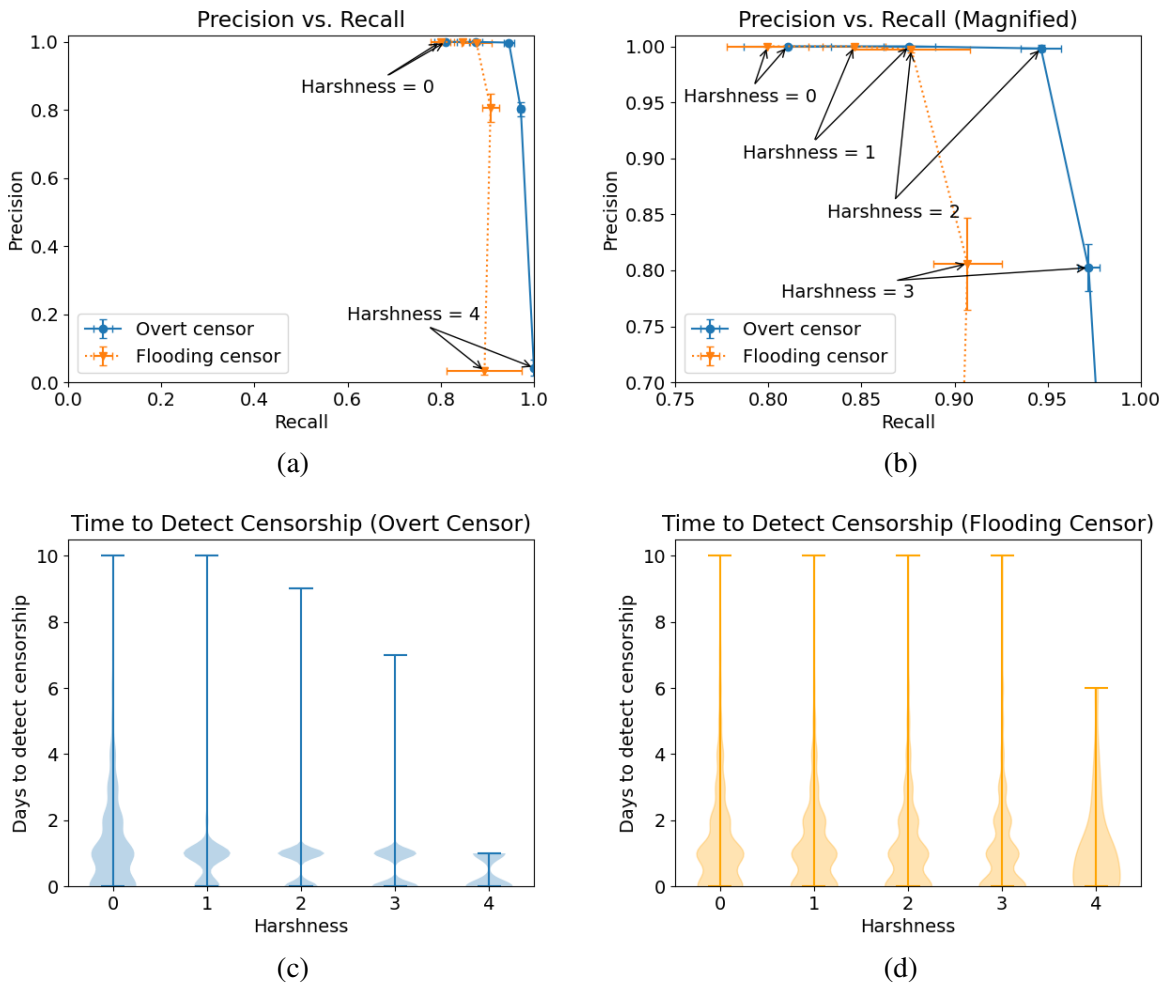


Figure 5.2: These plots show data from our second experiment. The probability of users submitting reports is held constant at 0.25, and the harshness of Troll Patrol’s classifier is varied.

Table 5.2: Results of the first experiment with the **overt censor**, specifically the mean and standard deviation number of true positives, true negatives, false positives, and false negatives for each set of trials. The independent variable in this experiment is the probability of users submitting reports.

Prob. users submit reports	True positives	True negatives	False positives	False negatives	Precision	Recall
0.0	410±30	13±4	1±1	36±3	0.999±0.002	0.92±0.01
0.01	430±20	20±10	2±2	28±4	0.996±0.004	0.94±0.01
0.05	400±100	10±10	2±1	30±10	0.996±0.002	0.93±0.02
0.1	410±10	13±5	2±1	30±10	0.995±0.002	0.93±0.02
0.15	460±40	22±4	2±1	30±10	0.996±0.002	0.95±0.02
0.2	420±30	20±10	1±1	19±5	0.998±0.002	0.96±0.01
0.25	490±30	20±10	2±2	24±4	0.997±0.003	0.95±0.01
0.3	490±20	20±10	1±2	16±5	0.997±0.004	0.97±0.01
0.35	470±20	22±5	0.6±0.5	20±10	0.999±0.001	0.96±0.01
0.4	480±50	30±10	1±1	10±10	0.998±0.003	0.98±0.01
0.45	510±40	20±10	1±1	12±3	0.997±0.002	0.98±0.01
0.5	510±20	30±10	1±1	20±10	0.998±0.002	0.97±0.01
0.6	500±40	20±4	1±1	9±4	0.999±0.002	0.98±0.01
0.7	490±40	20±10	1±1	10±10	0.998±0.001	0.98±0.01
0.8	520±40	30±10	0.4±0.5	7±3	0.999±0.001	0.99±0.01
0.9	500±100	23±5	1±1	6±4	0.998±0.002	0.99±0.01
1.0	520±40	20±10	2±2	6±3	0.997±0.003	0.99±0.01

Table 5.3: Results of the first experiment with the **flooding censor**, specifically the mean and standard deviation number of true positives, true negatives, false positives, and false negatives for each set of trials. The independent variable in this experiment is the probability of users submitting reports. When Troll Patrol does not detect that bridges are blocked, Lox does not allow users to migrate to new bridges, so the number of overall bridges in the simulation does not grow. This accounts for the low number of overall bridges when the number of positive classifications (both true and false) is low.

Prob. users submit reports	True positives	True negatives	False positives	False negatives	Precision	Recall
0.01	30±10	2±2	1±1	180±10	0.96±0.04	0.13±0.03
0.05	180±10	20±10	1±1	160±20	0.996±0.004	0.53±0.04
0.1	300±20	20±10	2±1	120±20	0.995±0.003	0.72±0.04
0.15	370±30	40±10	2±2	90±20	0.996±0.005	0.8±0.04
0.2	440±20	40±10	1±1	70±10	0.997±0.002	0.86±0.01
0.25	470±20	42±3	1±1	60±10	0.997±0.002	0.89±0.01
0.3	450±30	40±10	2±2	50±10	0.997±0.004	0.9±0.02
0.35	480±20	30±10	2±1	40±10	0.996±0.003	0.92±0.01
0.4	500±30	30±10	1±1	40±10	0.998±0.002	0.92±0.01
0.45	500±30	40±10	2±2	40±10	0.997±0.004	0.93±0.02
0.5	510±30	40±10	2±2	31±3	0.996±0.003	0.943±0.004
0.6	520±30	30±10	1±1	26±4	0.998±0.002	0.95±0.01
0.7	500±20	30±10	2±2	20±10	0.997±0.004	0.96±0.01
0.8	550±50	30±10	2±1	17±4	0.997±0.003	0.97±0.01
0.9	530±20	40±10	0±0	20±10	1.0±0.0	0.96±0.01
1.0	560±20	30±10	1±1	20±10	0.998±0.001	0.97±0.01

Table 5.4: Results of the second experiment with the **overt censor**, specifically the mean and standard deviation number of true positives, true negatives, false positives, and false negatives for each set of trials. The independent variable in this experiment is the harshness of the classifier.

Harshness	True positives	True negatives	False positives	False negatives	Precision	Recall
0	390±10	40±10	0±0	90±10	1.0±0.0	0.81±0.02
1	440±30	22±4	0±0	62±5	1.0±0.0	0.88±0.01
2	500±100	30±10	1±2	27±5	0.998±0.003	0.95±0.01
3	420±30	20±10	100±10	12±2	0.8±0.02	0.97±0.01
4	20±10	0±0	500±100	0±0	0.04±0.02	1.0±0.0

Table 5.5: Results of the second experiment with the **flooding censor**, specifically the mean and standard deviation number of true positives, true negatives, false positives, and false negatives for each set of trials. The independent variable in this experiment is the harshness of the classifier.

Harshness	True positives	True negatives	False positives	False negatives	Precision	Recall
0	370±20	40±10	0±0	90±10	1.0±0.0	0.8±0.02
1	410±20	30±10	0±0	80±10	1.0±0.0	0.85±0.02
2	440±30	40±10	1±1	60±20	0.997±0.003	0.88±0.03
3	430±20	30±10	100±20	40±10	0.81±0.04	0.91±0.02
4	20±10	0±0	570±40	2±2	0.03±0.01	0.9±0.1

in this plot, corresponding to Troll Patrol taking either 0 days or 1 day to detect censorship. When the probability of users submitting reports is 0, there is only one area with observable density at 1 day, but as more users submit reports, we can see that Troll Patrol is more likely to detect censorship on the same day that the bridge is blocked. As the probability of users submitting reports increases from 0 in Figures 5.1a and 5.1b, we also observe a gradual trend of recall improving (increasing from 0.92 ± 0.01 when no users submit reports to 0.99 ± 0.01 when all users submit reports, as reported in Table 5.2) while precision does not change significantly. This indicates that Troll Patrol is able to correctly detect more blockages if more users submit reports.

While negative reports were *helpful* for detecting censorship by an overt censor, we found that they were *essential* for detecting censorship by a flooding censor, something that is not possible with publicly reported bridge statistics alone. As we see in Figure 5.1c, when only 1% of users submit reports, Troll Patrol fails to detect almost all blockages by the flooding censor; however, when the probability of users submitting reports increases, Troll Patrol’s recall rapidly improves. We can see in Figure 5.1d that even in the presence of this flooding censor, Troll Patrol was able to identify over 85% of blocked bridges when only 25% of users submitted reports.

We observed that in all comparisons (in both Figure 5.1 and Figure 5.2), the flooding censor was better able to evade detection than the overt censor. By comparing Figure 5.1e to Figure 5.1f and Figure 5.2c to Figure 5.2d, we observe that Troll Patrol often took longer to detect censorship by the flooding censor than by the overt censor. While we consider bridge stats insufficient on their own for reliably detecting censorship, this finding highlights that they are still useful against a censor that does not manipulate connection counts. Thus, we do not suggest replacing bridge stats with negative reports; instead, the two should be used together. As positive reports never impacted Troll Patrol’s classifications in our experiments, we conclude that this type of report is unlikely to be useful in practice.

We now discuss the results of our second experiment. In terms of tuning our classifier, we found that using a harshness value of 2 produced favorable results. In this case, bridges should be considered blocked upon receipt of 3 or more valid negative reports in one day or if the reported connection count from a given country falls below either 16 or the maximum number of reports observed for that bridge in the past 30 days minus 16, whichever is lower. In the case that the bridge has been active for 30 days or fewer, it may be considered blocked if it reports 0 connections but has reported 16 or more connections in the past. We see in Figures 5.2a and 5.2b that lower harshness values (corresponding to a higher threshold for negative reports and a lower threshold for reported connection counts) saw a decrease in recall for only marginal precision gains, while higher values decreased precision significantly, with a harshness value of 4 resulting in a high number of false positives. Additionally, from Figure 5.2c, we can see that it took Troll Patrol longer to detect censorship than it did with a harshness value of 1 or higher.

5.4 Conclusion

In this chapter, we describe the setup of our experiment and provide its results. We discuss the implications of these results, concluding that negative reports are helpful for detecting bridge censorship, and we recommend incorporating negative reports, but not positive reports, into the Tor Project's bridge blockage detection system. In the next chapter, we will discuss shortcomings of this study and future directions that may improve upon our work.

Chapter 6

Limitations and Future Work

In this chapter, we identify shortcomings of Troll Patrol and the simulation we used to evaluate it, as well as areas where future work may extend or improve upon our contributions.

6.1 Limitations

6.1.1 Country-Level Assumptions

Troll Patrol considers whether or not people in a specific country are able to access a specific bridge. This reflects the state of other related systems. Bridge statistics, for example, report `bridge-ips` on a per-country basis, and `rdsys` stores whether or not bridges are blocked based on country codes [TP23a]. Whether or not a bridge is accessible from a given country is a reasonable question when the country in question implements censorship uniformly for all residents; however, this is not the case everywhere. In some countries, such as India, different ISPs implement different censorship policies [SGB20], so a bridge blocked for some users may be accessible to others. For the purpose of Lox, which assumes a bridge revealed to any censor should be considered blocked for all users, this granularity is acceptable, but it may make detecting blockages more difficult when censorship is not applied uniformly. User reports could provide more granular information about users, such as indicating specific regions within a country or specific ISPs; however, this would negatively impact user privacy. Perhaps a compromise, such as differentially private location reporting, could provide a reasonable compromise between the goals of accurate censorship detection and user privacy. (However, adoption of techniques such as differential privacy could also make it more difficult to explain the privacy implications of reports to users, resulting in lower rates of user consent to report submission.)

Additionally, we evaluate Troll Patrol in a simulated setting in which all users are located within the same country. Identifying censorship when the users of a bridge are spread out across more countries may be more challenging, as Troll Patrol would have less per-country data on bridge use. For this reason, we recommend that Lox distribute each open-entry bucket only to users in the same country when it is possible to infer users' countries. If each open-entry bucket is distributed primarily within a single country, Lox should select open-entry buckets that were distributed to users in the same country to group together in invite-only superset buckets. We note that it may not be possible to infer a user's country when they request an open invitation, and users may invite friends from other countries to join their buckets. Nevertheless, taking this step would help to ensure that Troll Patrol has enough data from a single country to evaluate whether or not a bridge is blocked in that country.

6.1.2 No Model of Social Graphs

In our simulation, we simplistically assume that each time an invitation is issued, it has a certain probability of being issued to a censor agent, and if it is issued to an honest user, it is essentially issued to a random honest user who needs an invitation. As we note, one implication of this is that given enough time, the censor is able to infiltrate most buckets. A more accurate way to model social graphs may be to give each user an individualized probability of unknowingly being friends with a censor agent. Additionally, first defining the actual social graphs between groups of users and having users only provide invitations to their friends (or possibly friends of friends), rather than random users, would be a more realistic way to model the trust networks leveraged by Lox. This would likely allow us to observe more accurate bridge censorship dynamics, improving our analysis.

6.2 Future Work

6.2.1 Detecting Mass-Blockage Events

The threat models of Lox and Troll Patrol assume that censors learn about individual bridges in the same ways honest users do; however, we recognize that censors may also be able to identify and block a large number of bridges at once through other means, for example by finding a flaw in a PT's protections against protocol-level fingerprinting or active probing, or by heuristically blocking fully encrypted traffic [WSS⁺23]. The Lox paper notes that if a mass-blockage event, such as a censor blocking all lyrebird bridges, were to occur, users should be given new bridges,

but some users may migrate before this happens, causing them to lose trust levels [TG23]. The Lox Distributor stores previous versions of its own state [Tu23], and in response to a mass-blockage event, it can roll back to some point in time prior to the event to allow users who lost trust to reuse their old credentials. A useful extension for Troll Patrol would be detecting such mass-blockage events and flagging them as special events that may require rollback action from Lox. Of course, a censor may simply learn many bridges but wait to block them until a specific time (such as during an election). Such an occurrence may appear similar to a censor suddenly fingerprinting and blocking a PT. In attempting to identify protocol-level blocking events, Troll Patrol should not absolve users of actual cooperation with censors.

6.2.2 Shorter Analysis Periods

We design Troll Patrol primarily to fulfill a requirement of Lox, and we tailor it to Lox's needs. As Lox currently requires knowledge of bridge blockages once per day, and published bridge statistics cover a 24-hour period, Troll Patrol collects data representing an entire day and uses this data to draw its conclusions. However, reducing the duration of this period may be useful for applications that need a faster answer about bridge reachability. One possible method for accelerating analysis would be determining the minimum number of negative reports that would result in the bridge being blocked and immediately reporting the blockage to the bridge's distributor upon receipt of that number. This would require Troll Patrol to send reports to the distributor for verification immediately after receiving them (or after receiving at least this threshold number), rather than at the end of the day.

6.2.3 Report Submission

We do not prescribe the client report submission process. We provide designs for the reports, and our reference implementation includes functions for creating the reports, but we assume that these functions will be called by Tor Browser, either automatically or manually. Here, we provide considerations for implementation of report submission in client software. We ultimately do not recommend the use of positive reports, so we focus only on the details of submitting negative reports.

Imperatively, reports must only be submitted if the user has consented to their submission. Ideally, reporting would be automated by Tor Browser. In this case, the user should be prompted either when they first open the browser or when they are first eligible to submit reports (i.e., when they are first unable to connect to a bridge). The latter may be a more effective strategy for informing users of the purpose of the reports so that they can give consent. One can easily

imagine a user dismissing a telemetry request out-of-hand upon starting the Tor Browser for the first time but being willing to report a known issue when they are unable to connect. Users should be able to consent or refuse to consent, and they should be able to change their decision at any time. Alternatively, the user may be prompted to submit a report each day they are unable to connect to a bridge. In this case, the user must be provided an option to disable these prompts altogether.

We specifically design reports to avoid revealing private user information. Negative reports identify that a connection to a bridge was attempted and failed, along with the following information:

- The hashed bridge fingerprint
- The user's country
- The date of the connection attempt
- The distributor that distributed the bridge
- Additional cryptographic information

These reports identify users' countries but nothing more precise. The additional cryptographic information is a hash of the bridge line or β (along with a nonce and date). No input to the hash relies on information about the user; these hashes only reflect information about bridges. In Section 6.2.5, we discuss replacing this hash with a Lox proof in order to restrict negative report submission. If this change is implemented, the cryptographic information will be a zero-knowledge proof that allows the user to prove that they have a Lox credential without revealing the credential's private fields. (The LA would be able to determine the user's bucket, i.e., information about the user's bridges, but no information that identifies the user or links the proof to other proofs.) In either case, this additional cryptographic information does not identify users with any greater precision than revealing the fact that they are legitimate users of the bridge.

If the client presents the exact pieces of (broadly) identifying data and how we will use the data to users (in an easy-to-understand format that does not require a nuanced understanding of the cryptography involved) when requesting consent, we believe that users will be more willing to grant consent, as they will better understand the very minimal privacy implications. We emphasize that privacy protections do not obviate the need for consent; even though we reasonably believe that the reports contain no personal information, users should not be forced to submit them. We consider that users may be motivated to submit negative reports when they are unable to connect but not positive reports when everything works well. It may be possible to entice

users to submit positive reports by offering some sort of reward, such as more Lox invitations to give to friends or shorter wait times before leveling up. However, given that our evaluation in Chapter 5 found no impact of positive reports whatsoever, it seems unlikely that positive reports will be useful enough to warrant incentives for users.

The timing of report submission must also be decided. Ideally, each user should submit reports each day they attempt to connect to their bridges. If the user submits each report immediately after their first connection attempt of the day, this leaks information about the connecting client's behavior to Troll Patrol. If the user's client submits all of its collected reports when it closes, this also leaks information. Additionally, if Troll Patrol's analysis period is shortened, this could delay the conclusion that a bridge is blocked. If the user waits a random amount of time to submit each report, we risk the client going offline before submitting some reports. One option would be for Tor Browser to submit reports either after a random delay or when it closes, whichever comes first.

These matters should be carefully considered when designing the reporting component of client software.

6.2.4 Negative Report Encryption Key Distribution

As described in Chapter 3, negative reports should be encrypted with a periodically rotated key. Troll Patrol should generate a long-term signing key and use this to sign each new encryption key. Then, the encryption key must be made available to all users in some way that is difficult or expensive for the censor to block. We do not prescribe this process, but we suggest in Section 3.2.2 that the current key could be distributed along with Lox's encrypted bridge table.

6.2.5 Limiting Negative Reports

As discussed in Chapter 3, we do not currently provide a technical restriction to the number of reports an eligible user can submit; however, we suggest an approach for limiting positive reports to one per bridge per credential per day. This approach could also be applied to negative reports if desired. This change would require a Lox proof to be added for negative reports, as it is for positive reports. The negative report must prove that the user has a Lox credential with the bucket containing the bridge being reported and that that credential has not yet been used to report that bridge on that date. This proof could safely replace the hash-based proof of bridge knowledge in the current negative report design. If Troll Patrol is also used with non-Lox bridge distributors, this restriction would only be implemented for Lox-distributed bridges, and Troll

Patrol would still need to accept hashes in reports for non-Lox bridges. In this case, Troll Patrol should reject negative reports that indicate Lox as the distributor but use hash-based proofs of bridge knowledge. We note that the use of this Lox proof would allow us to remove the nonce, obviating the need for encrypting negative reports for Lox-distributed bridges. However, we do not recommend removing the encryption requirement. Encryption may still be preferred in case it offers any meaningful privacy benefit either now or due to future changes, and negative reports for bridges distributed by other distributors should still be encrypted. It would be simpler to continue requiring encryption for all negative reports.

We observe that even with this restriction, in some cases it may be possible for one user to submit two negative reports for the same bridge: one using their untrusted credential with the open-entry bucket value and the other using their trusted credential with the invite-only bucket value. However, this would only be possible if the user originally joined Lox using an open invitation and is reporting their original bridge. We also note that limiting reports in the way we describe would not prevent users from issuing invitations to themselves in order to obtain more Lox credentials and submit more negative reports. In Section 3.3.2 we discussed the possibility of a censor issuing invitations to itself in order to submit more positive reports and highlighted the fact that the censor must wait until each credential is level 3 before using it to submit these reports. Unlike positive reports, valid negative reports can be submitted by users of any level, so the barrier to submitting multiple negative reports for one bridge on one day is much lower.

As we discussed in Section 3.2, we are not particularly concerned with the possibility of censors falsely submitting negative reports, as we expect that a censor wishing to disrupt the use of the bridge will actually block the bridge. However, we consider the possibility that a troublemaker not affiliated with any censor learns bridges and submits negative reports for them so that Troll Patrol falsely detects that they are blocked. Restricting negative reports based on Lox credentials would not slow down such a troublemaker significantly. Thus, this restriction may not be useful, or it may only be useful for preventing accidental duplicate report submissions, e.g., due to software bugs.

We note that Lox is already equipped to handle malicious users who submit many negative reports; it will deal with them the same way it deals with censors. Their bridges will be detected as blocked, and unless they are able to migrate to new buckets, they will not be able to learn new bridges. Even if such a user is willing to be patient and waits to submit their reports until they have a level 3 credential, they will only be able to migrate a few times before they lose the ability to level up and with it the ability to continue migrating.

6.2.6 Active Scans

We do not implement active (direct, reverse, or indirect) scans as part of our system, but we suggest that Troll Patrol may be extended with this functionality, either to verify its conclusions or to gather initial ground truth for the purpose of developing a more accurate classifier. Including these techniques would require providing the system with sensitive data (i.e., the bridge lines to test) we have thus far designed it not to need. However, this compromise may be acceptable, especially as we consider that Troll Patrol may run on the same machine as the Lox Authority.

6.2.7 Evaluation with Additional Types of Censors

In Section 4.5, we discuss multiple approaches to censors' speed, secrecy, and totality, but our evaluation in Chapter 5 uses only two types of censors: overt and flooding censors that are both fast and full. Further evaluation of Troll Patrol's ability to detect censorship by other types of censors would be valuable.

Chapter 7

Conclusion

In this thesis, we showed that it is possible to design a bridge blockage detection system that harnesses privacy-preserving user reports in conjunction with bridge usage statistics to successfully detect more blocked bridges than would be detected using only bridge usage statistics. We showed that these user reports can allow such a system to detect censorship even when the censor takes measures to hide it.

To show this, we presented Troll Patrol, such a system for detecting censorship of Tor bridges. A central component of Troll Patrol is an anonymous reporting system that allows honest users to submit reports of blocked bridges without identifying themselves while preventing censors aiming to disrupt the system from submitting valid reports for bridges they do not know. Troll Patrol evaluates bridge reachability based on extant bridge usage statistics and these novel anonymous reports. In simulations, we found that Troll Patrol was able to accurately detect the majority of blocked bridges, and its accuracy improved when users submitted reports that bridges were blocked. These reports of blocked bridges also enabled Troll Patrol to defend against an attack it otherwise could not, in which the censor artificially inflated user connection counts to evade detection. We found that even when the censor employed this attack, Troll Patrol was able to detect over 85% of blocked bridges as long as at least 25% of users submitted reports.

It is our hope that our work will help the Tor Project test and deploy Lox to enable more people around the world to gain privacy, anonymity, and autonomy through the use of Tor.

References

- [ant98] antirez. new tcp scan method, 1998. Accessed August 2024. URL: <https://seclists.org/bugtraq/1998/Dec/79>.
- [BBF⁺24] Cecylia Bocovich, Arlo Breault, David Fifield, Serene, and Xiaokang Wang. Snowflake, a censorship circumvention system using temporary WebRTC proxies. In *Proceedings of the 33rd USENIX Security Symposium*, pages 2635–2652, Philadelphia, PA, August 2024.
- [BHKL13] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, pages 967–980, Berlin, Germany, November 2013.
- [Boc21a] Cecylia Bocovich. Possible new blocking event in Belarus, 2021. Accessed August 2024. URL: <https://gitlab.torproject.org/tpo/anti-censorship/censorship-analysis/-/issues/40002>.
- [Boc21b] Cecylia Bocovich. Tor blocking events in Belarus from 2020-2021, 2021. Accessed August 2024. URL: <https://gitlab.torproject.org/tpo/anti-censorship/censorship-analysis/-/blob/main/reports/2020/belarus/2020-belarus-report.md>.
- [BSR17] Diogo Barradas, Nuno Santos, and Luís Rodrigues. DeltaShaper: Enabling Unobservable Censorship-resistant TCP Tunneling over Videoconferencing Streams. *Proceedings on Privacy Enhancing Technologies*, 2017(4):1–18, 2017.
- [BSRN20] Diogo Barradas, Nuno Santos, Luís Rodrigues, and Vítor Nunes. Poking a Hole in the Wall: Efficient Censorship-Resistant Internet Communications by Parasitizing on WebRTC. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 35–48, Virtual Event, November 2020.

- [CMZ14] Melissa Chase, Sarah Meiklejohn, and Gregory M. Zaverucha. Algebraic MACs and Keyed-Verification Anonymous Credentials. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1205–1216, Scottsdale, AZ, USA, November 2014.
- [Dav15] David Fifield and Chang Lan and Rod Hynes and Percy Wegmann and Vern Paxson. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies*, 2015(2):46–64, 2015.
- [Din11a] Roger Dingledine. Five ways to test bridge reachability. Technical Report 2011-12-001, The Tor Project, December 2011. URL: <https://research.torproject.org/techreports/five-ways-test-bridge-reachability-2011-12-01.pdf>.
- [Din11b] Roger Dingledine. Ten ways to discover Tor bridges. Technical Report 2011-10-002, The Tor Project, October 2011. URL: <https://research.torproject.org/techreports/five-ways-test-bridge-reachability-2011-12-01.pdf>.
- [DM06] Roger Dingledine and Nick Mathewson. Design of a blocking-resistant anonymity system. Technical Report 2006-11-001, The Tor Project, November 2006. URL: <https://research.torproject.org/techreports/blocking-2006-11.pdf>.
- [DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, San Diego, CA, August 2004.
- [DRPC16] Frederick Douglas, Rorshach, Weiyang Pan, and Matthew Caesar. Salmon: Robust Proxy Distribution for Censorship Circumvention. *Proceedings on Privacy Enhancing Technologies*, 2016(4):4–20, 2016.
- [EFW⁺15] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas C. Weaver, and Vern Paxson. Examining How the Great Firewall Discovers Hidden Circumvention Servers. In *Proceedings of the 2015 Internet Measurement Conference*, pages 445–458, Tokyo, Japan, October 2015.
- [EKAC14] Roya Ensafi, Jeffrey Knockel, Geoffrey Alexander, and Jedidiah R. Crandall. Detecting Intentional Packet Drops on the Internet via TCP/IP Side Channels. In *Proceedings of the 15th International Conference on Passive and Active Measurement*, pages 109–118, Los Angeles, CA, USA, March 2014.

- [EPKC10] Roya Ensafi, Jong Chun Park, Deepak Kapur, and Jedidiah R. Crandall. Idle Port Scanning and Non-interference Analysis of Network Protocol Stacks Using Model Checking. In *Proceedings of the 19th USENIX Security Symposium*, pages 257–272, Baltimore, MD, August 2010.
- [FBS22] Gabriel Figueira, Diogo Barradas, and Nuno Santos. Stegozoa: Enhancing We-bRTC Covert Channels with Video Steganography for Internet Censorship Circumvention. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 1154–1167, Nagasaki, Japan, May 2022.
- [FHE⁺12] David Fifield, Nate Hardison, Jonathan D. Ellithorpe, Emily Stark, Dan Boneh, Roger Dingledine, and Phillip A. Porras. Evading Censorship with Browser-Based Proxies. In *Proceedings of the 12th International Symposium on Privacy Enhancing Technologies*, pages 239–258, Vigo, Spain, July 2012.
- [FJ23] Ellis Fenske and Aaron Johnson. Security Notions for Fully Encrypted Protocols. *Free and Open Communications on the Internet*, 2023(1):24–29, 2023.
- [FW20] Sergey Frolov and Eric Wustrow. HTTPPT: A Probe-Resistant Proxy. In *Proceedings of the 10th USENIX Workshop on Free and Open Communications on the Internet*, Virtual Event, August 2020.
- [ggu21] ggus. Responding to Tor censorship in Russia, 2021. Accessed August 2024. URL: <https://blog.torproject.org/tor-censorship-in-russia/>.
- [Gus21] Gustavo Gus. [Russia] Monitoring dynamic bridges health, 2021. Accessed August 2024. URL: <https://gitlab.torproject.org/tpo/community/support/-/issues/40054>.
- [HNGJ16] Bridger Hahn, Rishab Nithyanand, Phillipa Gill, and Rob Johnson. Games without Frontiers: Investigating Video Games as a Covert Channel. In *Proceedings of the 2016 IEEE European Symposium on Security and Privacy*, pages 63–77, Saarbrücken, Germany, March 2016.
- [JF15] Ben Jones and Nick Feamster. Can Censorship Measurements Be Safe(r)? In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, Philadelphia, PA, USA, November 2015.
- [LdV17] Isis Agora Lovecruft and Henry de Valence. Hyphae: Social secret sharing, 2017. URL: <https://patternsinthevoid.net/hyphae/hyphae.pdf>.

- [Lei10] Bruce Leidl. Obfuscated OpenSSH, 2010. Accessed August 2024. URL: <https://github.com/brl/obfuscated-openssh>.
- [Loe11] Karsten Loesing. Case study: Learning whether a Tor bridge is blocked by looking at its aggregate usage statistics: Part One. Technical Report 2011-09-002, The Tor Project, September 2011. URL: <https://research.torproject.org/techreports/blocking-2011-09-15.pdf>.
- [MLDG12] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. SkypeMorph: protocol obfuscation for Tor bridges. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pages 97–108, Raleigh, NC, USA, October 2012.
- [MML11] Damon McCoy, Jose Andre Morales, and Kirill Levchenko. Proximax: A Measurement Based System for Proxies Dissemination. In *Proceedings of the 15th International Conference on Financial Cryptography and Data Security*, pages 260–267, Gros Islet, St. Lucia, March 2011.
- [MTC17] Srdjan Matic, Carmela Troncoso, and Juan Caballero. Dissecting Tor Bridges: A Security Evaluation of their Private and Public Infrastructures. In *Proceedings of the 24th Network and Distributed System Security Symposium*, San Diego, CA, USA, February 2017.
- [NCW⁺20] Arian Akhavan Niaki, Shinyoung Cho, Zachary Weinberg, Nguyen Phong Hoang, Abbas Razaghpanah, Nicolas Christin, and Phillipa Gill. ICLab: A Global, Longitudinal Internet Censorship Measurement Platform. In *Proceedings of the 2020 IEEE Symposium on Security and Privacy*, pages 135–151, Virtual Event, May 2020.
- [Odd21] Space Oddity. [tor-relays] [Censorship in Russia] More of my bridges got blocked, 2021. Accessed August 2024. URL: <https://forum.torproject.org/t/tor-relays-censorship-in-russia-more-of-my-bridges-got-blocked/1518>.
- [PEL⁺17] Paul Pearce, Roya Ensafi, Frank H. Li, Nick Feamster, and Vern Paxson. Augur: Internet-Wide Detection of Connectivity Disruptions. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy*, pages 427–443, San Jose, CA, USA, May 2017.
- [pho11] phobos. New Blocking Activity from Iran, 2011. Accessed August 2024. URL: <https://blog.torproject.org/new-blocking-activity-iran/>.

- [pho12a] phobos. Iran partially blocks encrypted network traffic, February 2012. Accessed August 2024. URL: <https://blog.torproject.org/iran-partially-blocks-encrypted-network-traffic/>.
- [pho12b] phobos. Kazakhstan upgrades censorship to deep packet inspection, February 2012. Accessed August 2024. URL: <https://blog.torproject.org/kazakhstan-upgrades-censorship-deep-packet-inspection/>.
- [RSD⁺20] Ram Sundara Raman, Adrian Stoll, Jakub Dalek, Reethika Ramesh, Will Scott, and Roya Ensafi. Measuring the Deployment of Network Censorship Filters at Global Scale. In *Proceedings of the 27th Network and Distributed System Security Symposium*, San Diego, CA, USA, February 2020.
- [Run12] Runa. Ethiopia Introduces Deep Packet Inspection, May 2012. Accessed August 2024. URL: <https://blog.torproject.org/ethiopia-introduces-deep-packet-inspection/>.
- [Sc24] The Shadowsocks contributors. ShadowSocks, 2024. Accessed August 2024. URL: <https://github.com/shadowsocks>.
- [sg24] shelikhoo and ggus. Hiding in plain sight: Introducing WebTunnel, March 2024. Accessed August 2024. URL: <https://blog.torproject.org/introducing-webtunnel-evading-censorship-by-hiding-in-plain-sight/>.
- [SGB20] Kushagra Singh, Gurshabad Grover, and Varun Bansal. How India Censors the Web. In *Proceedings of the 12th ACM Conference on Web Science*, pages 21–28, Southampton, United Kingdom, July 2020.
- [SIWR22] Paul Schmitt, Jana Iyengar, Christopher Wood, and Barath Raghavan. The Decoupling Principle: A Practical Privacy Framework. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pages 213–220, Austin, TX, USA, November 2022.
- [SJP⁺11] Rob Smits, Divam Jain, Sarah Pidcock, Ian Goldberg, and Urs Hengartner. BridgeSPA: improving Tor bridges with single packet authorization. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 93–102, Chicago, IL, USA, October 2011.

- [TG23] Lindsey Tulloch and Ian Goldberg. Lox: Protecting the Social Graph in Bridge Distribution. *Proceedings on Privacy Enhancing Technologies*, 2023(1):494–509, 2023.
- [TP10] The Tor Project. Learn which bridges can't reach baidu, 2010. Accessed August 2024. URL: <https://gitlab.torproject.org/legacy/trac/-/issues/1851>.
- [TP11a] The Tor Project. Can we try to extend from the bridge to a website and learn if the website is reachable?, 2011. Accessed August 2024. URL: <https://gitlab.torproject.org/legacy/trac/-/issues/2576>.
- [TP11b] The Tor Project. GFW probes based on Tor's SSL cipher list, December 2011. Accessed August 2024. URL: <https://gitlab.torproject.org/tpo/core/tor/-/issues/4744>.
- [TP17] The Tor Project. Evaluate, possibly revise, and then implement ideas for TLS certificate normalization, 2017. Accessed August 2024. URL: <https://gitlab.torproject.org/tpo/core/tor/-/issues/7145>.
- [TP20] The Tor Project. Reachability tests for new obfs4 bridges, 2020. Accessed August 2024. URL: <https://gitlab.torproject.org/tpo/anti-censorship/pluggable-transport/lyrebird/-/issues/31701>.
- [TP22] The Tor Project. detect if a bridge is blocked in a certain country by its reported usage, 2022. Accessed August 2024. URL: <https://gitlab.torproject.org/tpo/anti-censorship/rdsys/-/issues/112>.
- [TP23a] The Tor Project. Backend API Documentation for Distributors, 2023. Accessed August 2024. URL: <https://gitlab.torproject.org/tpo/anti-censorship/rdsys/-/blob/eb46b076b47f698e05e6d9b6b3aa020c43d6633c/doc/backend-api.md>.
- [TP23b] The Tor Project. Brainstorm and analyze heuristics to guess that a bridge might be offline or blocked, 2023. Accessed August 2024. URL: <https://gitlab.torproject.org/tpo/anti-censorship/censorship-analysis/-/issues/40035>.

- [TP24a] The Tor Project. BridgeDB specification, 2024. Accessed August 2024. URL: <https://spec.torproject.org/bridgedb-spec>.
- [TP24b] The Tor Project. Bridges, 2024. Accessed August 2024. URL: <https://tb-manual.torproject.org/bridges/>.
- [TP24c] The Tor Project. Circumvention, 2024. Accessed August 2024. URL: <https://tb-manual.torproject.org/circumvention/>.
- [TP24d] The Tor Project. CollecTor, 2024. Accessed August 2024. URL: <https://metrics.torproject.org/collector.html>.
- [TP24e] The Tor Project. Consensus Health, 2024. Accessed August 2024. URL: <https://consensus-health.torproject.org/>.
- [TP24f] The Tor Project. Extra-info document format, 2024. Accessed August 2024. URL: <https://spec.torproject.org/dir-spec/extra-info-document-format.html>.
- [TP24g] The Tor Project. Let bridge users choose to only reach their first working bridge, 2024. Accessed August 2024. URL: <https://gitlab.torproject.org/tpo/core/tor/-/issues/40578>.
- [TP24h] The Tor Project. Lox: A privacy preserving bridge distribution system, 2024. Accessed August 2024. URL: <https://gitlab.torproject.org/tpo/anti-censorship/lox/>.
- [TP24i] The Tor Project. lyrebird, 2024. Accessed August 2024. URL: <https://gitlab.torproject.org/tpo/anti-censorship/pluggable-transport/lyrebird/>.
- [TP24j] The Tor Project. moat, 2024. Accessed August 2024. URL: <https://support.torproject.org/glossary/moat/>.
- [TP24k] The Tor Project. Pluggable Transport Specification (Version 1), 2024. Accessed August 2024. URL: <https://spec.torproject.org/pt-spec/>.
- [TP24l] The Tor Project. Rdsys, 2024. Accessed August 2024. URL: <https://gitlab.torproject.org/tpo/anti-censorship/rdsys/>.
- [Tul22] Lindsey Tulloch. Lox: Protecting the Social Graph in Bridge Distribution. Master's thesis, University of Waterloo, 2022.

- [Tul23] Lindsey Tulloch. Store Lox bridge table and spent credential tables in a disk-backed database, 2023. Accessed August 2024. URL: <https://gitlab.torproject.org/tpo/anti-censorship/lox/-/issues/21>.
- [twi12] twilde. Knock Knock Knockin’ on Bridges’ Doors, January 2012. Accessed August 2024. URL: <https://blog.torproject.org/knock-knock-knockin-bridges-doors/>.
- [VMS⁺18] Benjamin VanderSloot, Allison McDonald, Will Scott, J. Alex Halderman, and Roya Ensafi. Quack: Scalable Remote Measurement of Application-Layer Censorship. In *Proceedings of the 27th USENIX Security Symposium*, pages 187–202, Baltimore, MD, August 2018.
- [WCC⁺18] Zachary Weinberg, Shinyoung Cho, Nicolas Christin, Vyas Sekar, and Phillipa Gill. How to Catch when Proxies Lie: Verifying the Physical Locations of Network Proxies with Active Geolocation. In *Proceedings of the Internet Measurement Conference 2018*, pages 203–217, Boston, MA, USA, October 2018.
- [WL12] Philipp Winter and Stefan Lindskog. How the Great Firewall of China is Blocking Tor. In *2nd USENIX Workshop on Free and Open Communications on the Internet*, Bellevue, WA, August 2012.
- [WLBH13] Qiyan Wang, Zi Lin, Nikita Borisov, and Nicholas Hopper. rBridge: User Reputation based Tor Bridge Distribution with Privacy Preservation. In *Proceedings of the 20th Network and Distributed System Security Symposium*, San Diego, CA, USA, February 2013.
- [WPF13] Philipp Winter, Tobias Pulls, and Jürgen Fuß. ScrambleSuit: a polymorphic network protocol to circumvent censorship. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 213–224, Berlin, Germany, November 2013.
- [WSS⁺23] Mingshi Wu, Jackson Sippe, Danesh Sivakumar, Jack Burg, Peter Anderson, Xiaokang Wang, Kevin Bock, Amir Houmansadr, Dave Levin, and Eric Wustrow. How the great firewall of china detects and blocks fully encrypted traffic. In *Proceedings of the 32nd USENIX Security Symposium*, pages 2653–2670, Anaheim, CA, August 2023.
- [WWY⁺12] Zachary Weinberg, Jeffrey B. Wang, Vinod Yegneswaran, Linda Briesemeister, Steven W. T. Cheung, Frank Wang, and Dan Boneh. StegoTorus: a camouflage

proxy for the Tor anonymity system. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pages 109–120, Raleigh, NC, USA, October 2012.

- [XRS⁺21] Diwen Xue, Reethika Ramesh, Valdik S S, Leonid Evdokimov, Andrey Viktorov, Arham Jain, Eric Wustrow, Simone Basso, and Roya Ensafi. Throttling Twitter: an emerging censorship technique in Russia. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 435–443, Virtual Event, November 2021.
- [ZMW⁺20] Pengxiong Zhu, Keyu Man, Zhongjie Wang, Zhiyun Qian, Roya Ensafi, J. Alex Halderman, and Haixin Duan. Characterizing Transnational Internet Performance and the Great Bottleneck of China. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(1), May 2020.

APPENDICES

Appendix A

Source Code Availability

This work produced the following Rust projects, all publicly available and licensed under the MIT/Expat license.

- <https://git-crysp.uwaterloo.ca/vvecna/troll-patrol> – This is the main codebase for Troll Patrol.
- <https://git-crysp.uwaterloo.ca/vvecna/lox-simulation> – This is the simulation. We include a Dockerfile for reproducing our simulations.
- https://git-crysp.uwaterloo.ca/vvecna/lox_cli – This is a simple Lox client library and command-line interface. It is used to perform user actions in our simulation.

Additionally, we made modifications to the Lox codebase. Our fork of the Lox project is available at <https://gitlab.torproject.org/vecna/lox>.