The Unbalancing Act

Proxy Preservation for Censorship Resistance Systems

by

Jodi Spacek

Bachelor of Computer Science, UBC, 2014

Bachelor of Music, Mount Allison University, 2000

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL
STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

April 2019

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

The Unbalancing Act

submitted by **Jodi Spacek** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

Examining Committee:

Ivan Beschastnikh, Computer Science
Supervisor

Norman Hutchinson, Computer Science
Supervisory Committee Member

Abstract

Internet censorship is a form of digital authoritarianism in certain countries that restrict access to the internet. Internet freedom, to a degree, is possible even in these countries by means of proxies maintained outside of the censor's boundaries. These proxies can be compromised by censors who pose as legitimate users to discover proxies. Censors are powerful adversaries and may block access to any proxy once they know about it.

We propose a novel technique to address the proxy distribution problem in this thesis. We introduce the *needle* algorithm that preserves proxies by limiting their distribution. We show that it is a useful mechanism for both preserving proxies and maintaining client service under a censorship threat model.

We examine characteristics of the needle algorithm in a simulation. Three measures are important under the censorship threat model; the enumeration or discovery of all proxies, load balancing guarantees, and the collateral damage of innocent bystanders. We compare the results of these experiments with two well-known algorithms, uniform random and power of 2 choices, as well as Tor's `bridgedb` proxy assignment mechanism.

Lay Summary

Internet censorship prevents citizens of a country from accessing content on specific websites. Censors are powerful figures that restrict free access to information by limiting their citizens' internet access when they are within the censor's country. A proxy is a single hop beyond the reach of the censor that is controlled by an organization on the outside. A censored user connects to the proxy to hop outside of their country to view restricted content. There are not enough proxies for each censored user to have their own, so they must share. A censor may pretend to be honest to learn about the proxies to block access to them. This thesis presents an approach to proxy distribution that preserves some proxies by restricting their distribution. We examine the lifetime of proxies in a simulation to see if this solution is practical in a real-life setting.

Preface

The original and unpublished work presented in this thesis was conducted by the author in the Networks, Systems, and Security (NSS) lab in the Department of Computer Science at the University of British Columbia (UBC) in Vancouver. It was designed, documented, and written with assistance from Dr. Ivan Beschastnikh, Dr. Norman Hutchinson, and Dr. William Aiello. Figures 1.1, 1.2, and 1.3 were created by Carlo Cossette. All other figures, tables, and experiments were designed and conducted by the author.

Table of Contents

List of Tables

List of Figures

Glossary

CRS	Censorship Resistance Systems
CCP	Coupon Collector Problem
CCP-R	Coupon Collector Problem Ring
OONI	Open Observatory of Network Interference
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
DNS	Domain Name System
IP	Internet Protocol
CDN	Content Delivery Networks
ISP	Internet Service Providers

Acknowledgements

This research was funded by a Canada Graduate Scholarships-Master's (CGS-M) grant from the Natural Science and Engineering Research Council of Canada (NSERC).

I would like to thank my advisor, Dr. Ivan Beschastnikh for his patience and guidance through the trials of learning how to research. I would also like to thank Dr. Norman Hutchinson for his insights, and Dr. William Aiello for helping me to carve out this research topic.

To my partner, Carlo, whose support through this endeavour has been unwavering.

To my grandmother, Marjorie Špaček, thank you for teaching me how to code on your Commodore 64. To my grandfather, Dr. Miroslav Špaček, for teaching me to question everything. And to my great uncle, RNDr. Ladislav (Adus) Špaček, for inspiring me through music and math.

A special thanks to all my teachers for providing answers to my questions.

Chapter 1

Introduction

Internet censorship is a significant and growing problem that threatens our freedom of expression and access to information. A citizen within a censored country attempts to access the internet that is hosted beyond the censor's control boundary, shown in Figure 1.1. A censor is a strong nation state adversary that conducts mass surveillance and utilizes blacklists. One of the simplest techniques employed by censors is to deny access to users by blocking traffic destined for blacklisted sites.

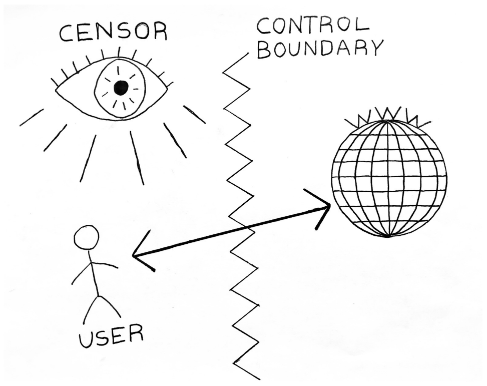

Figure 1.1: A censor observes a citizen's network activity.

This effective censorship technique can be circumvented by means of proxies; single-hop servers outside of the censored country that facilitate indirect access to censored information. In Figure 1.2, a citizen is blocked from the internet but manages to access sites via a proxy. Censorship Resistance Systems (CRS) like rBridge [25], meek [9], and Tor's `bridgedb` [2] manage proxies and allow users in censored countries to access blacklisted websites. These systems bypass censors and route users through secret paths. Secret paths rely on proxies that are managed by censorship resistance systems. Proxies serve as intermediary hops between a user and a blacklisted destination site.

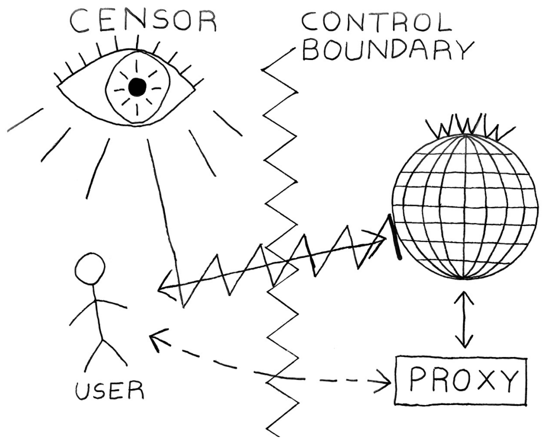

Figure 1.2: The citizen uses a proxy to access the internet.

CRS are composed of honest and dishonest (fake) users, several honest proxies, and a centralized proxy distributor that is also honest. Figure 1.3 shows a typical censorship circumvention system using a collection of proxies and a centralized distributor. CRS rely on the proxy distributor to assign clients to proxies. Honest users anonymously request proxies and receive proxy information details from the distributor. However, a censor may also learn proxy details by posing as an honest user via legitimate, anonymous methods. The censor coordinates information collected through a collection of fake accounts to gain knowledge of the proxies in the circumvention system.

To get an idea of the scale of censorship events around the world, Figure 1.4 shows a world map where the Open Observatory of Network Interference (OONI) project's network measurement tests, `ooniprobes`, are run daily by volunteers [3]. The red areas on the map outline confirmed cases of censorship. Censorship is confirmed by the response body of a Hypertext Transfer Protocol (HTTP) request of a blocked page from within a censored country.

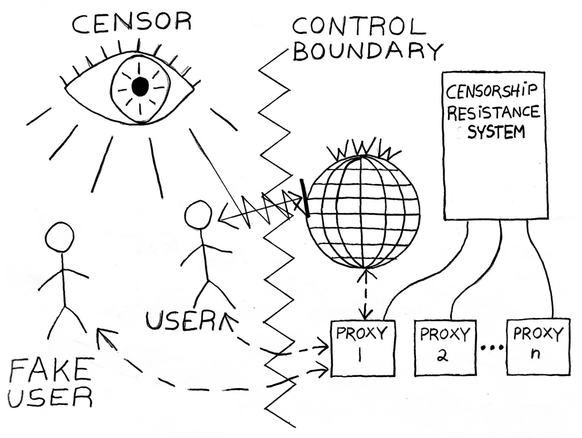

Figure 1.3: Censorship circumvention via proxy distribution.

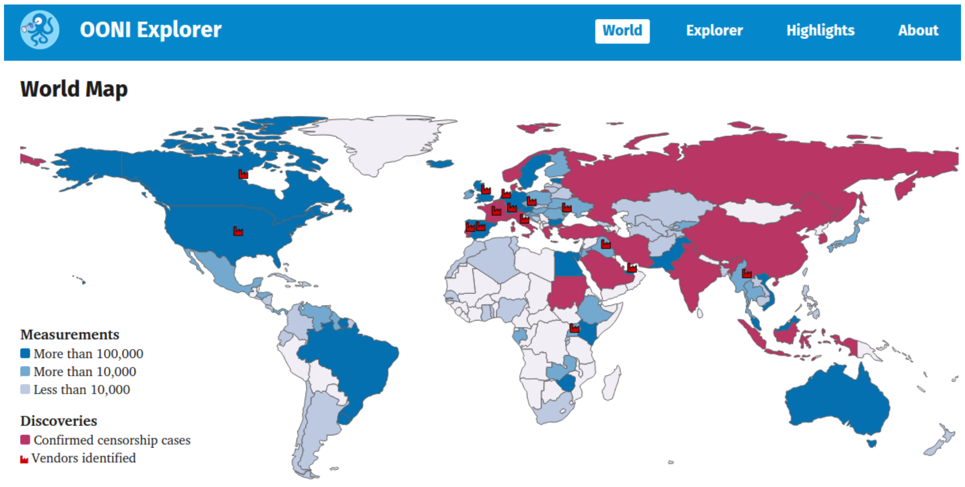

Figure 1.4: OONI project's censorship data.

Proxies in censorship resistance systems that cannot distinguish between honest users and fake users are destined for discovery over time. Most systems manually *reserve* a set of proxies that is only distributed when all other proxies are compromised. This manual solution is effective, but does not maximize the allocation of proxies.

We offer a lightweight, elegant solution to the proxy distribution problem by slowing down the progress of the censor to learn new proxies. Our goal is to provide service to clients while *preserving* proxies from distribution, a goal that is complimentary to *reserving* proxies. We introduce a partially random strategy for the proxy distribution problem to distribute proxies to honest clients while fake clients are present. Our approach uses principles of load balancing in a non-intuitive way to preserve random proxies by unbalancing the load on proxies, making them more difficult to discover.

We provide an approximation of the censor's required effort to discover all of the proxies in a system based on the coupon collector problem [11]. We validate this approach in a simulator that simulates a proxy distributor in a censorship resistance system.

We contribute to the existing body of knowledge in the following five ways:

1. We introduce a novel algorithm, *needle*, that restricts the distribution of proxies in order to slow the expected time of proxy system compromise while still maintaining service to a proportion of clients.

2. We define a censorship threat model within which we evaluate the needle algorithm. We define honest users, a proxy distributor, insider attackers, popularity blocking, and a colluding censor within this model.

3. We analyze the needle algorithm in the context of the coupon collector problem to give bounds on the average time to collect all of the proxies.

4. We build a simulator to evaluate the needle proxy distribution algorithm and compare the results with Tor's `bridgedb` distribution, uniform random distribution, and the power of 2 choice load balancing algorithms.

5. We discuss the benefits and drawbacks of trust-based proxy distribution vs. lightweight approaches and provide ideas for future work on our approach.

Chapter 2

Background

2.1 Censorship Threat Model

The first goal of our censor in this model is to discover proxies. The censor discovers, or *enumerates*, proxies by posing as a legitimate user of a CRS. The distribution of proxies to honest users and the censor posing as an honest user is a cat-and-mouse game between the censor and a CRS, referred to as the *proxy distribution problem*.

There are three attacks that lead to the ability of our censor to block a proxy; the Sybil attack followed by the insider attack leading to the enumeration attack. In the first stage of the insider attack, a malicious entity, such as a censor, deploys multiple accounts through anonymous authentication. This is the *Sybil attack* that creates fake accounts to gain entry to the system. The *insider attack* occurs when the knowledge of assigned proxies gained from honest proxy assignment is given to the censor. The *enumeration attack* that follows is usually the action of the censor to block all of the proxies it knows about [25].

Once a censor knows about a proxy, it can decide to block that proxy. The second goal of our censor is to block the largest amount of honest users as possible. The censor may choose to block immediately when a proxy is discovered, known as *immediate blocking*. The censor may instead *delay blocking* in order to maximize the collateral damage to honest clients, or perhaps wait for a critical time or maximum number of users to time a blocking attack.

In addition, the censor keeps track of proxies that are the most *popular*. The censor determines that a proxy is popular by the number of times a proxy is assigned compared to other proxies that it knows about. The censor then chooses to block proxies based on their popularity ranking.

We make different kinds of blocking assumptions in the evaluation section in Chapter 4. Immediate blocking makes the most sense when we evaluate enumeration because as soon as the censor is assigned to any proxy, it is enumerated. We assume delayed blocking behaviour in the load balancing analysis because we want to observe load balancing trends over time, without any blocking on the part of the censor. The bystander evaluation assumes that the censor blocks the most popular proxies so that we can measure the number of bystanders in the remaining proxies.

2.1.1 Out of Scope

The censor is a powerful, nation-state adversary and many of its capabilities are beyond the scope of this work. For example, network-level proxy discovery where the censor actively or passively scans Internet Protocol (IP) addresses is not addressed. A censor may monitor all traffic in their network, store the state of the network, and monitor requests and responses to known proxy addresses. This allows the censor to potentially mount a zig-zag attack to connect users together by their proxies [1]. Only proxy discovery by assignment of an attacker to a proxy is explored in this thesis.

2.2 Coupon Collector Problem

In our model, the problem of proxy distribution is simply to distribute a collection of n proxies to some users, where k of these users are attackers, in such a way as to provide a degree of functionality to the honest users. From the censor's viewpoint, it closely aligns to the Coupon Collector Problem (CCP) where the proxies are coupons and the censor is the coupon collector.

CCP is a classic occupancy problem that considers the number of coupons that must be collected before one has the entire collection of coupons. There are n distinct coupons that are collected one at a time where the i^{th} coupon arrives with probability p_i [21]. The uniform random algorithm, shown in Algorithm 1, chooses one coupon randomly from the list of total coupons.

Where probabilities p_i from $i = 1$ to n are equal, the arrivals are uniform random. This equi-probable case, where all probabilities of coupons are equal, shows that the probability of obtaining a new coupon i decreases with each draw. The average number of draws to collect all of the coupons is nH_n where H_n is the n^{th} harmonic number [11]. We briefly show why this is the case below.

The summation of probabilities of obtaining a distinct coupon, not previously seen, is $\frac{n}{n} + \frac{n-1}{n} + \frac{n-2}{n} + ... + \frac{1}{n}$. The probability of receiving a unique coupon decreases by 1 in the numerator for each selection. The probability of a new coupon in the first draw is 100% for instance, because no coupon has yet appeared. The very last coupon is the most difficult to select with probability = $1/n$. By linearity of expectations, where p_i is the probability of obtaining the i^{th} coupon, this can be written as:

$$p_i = \sum_{i=0}^{n-1} \frac{n-i}{n}$$

This follows a geometric distribution, therefore $E[X_i] = \frac{1}{p_i}$ where X is the sum of the expected number of draws for the i^{th} coupon from i to n. The expected number of draws for all n coupons in total is:

$$E[X] = \sum_{i=0}^{n-1} \frac{n}{n-i}$$

By changing the variable i to $j = n - i$, we obtain $E[X] = nH_n$, where H_n is the n^{th} harmonic number:

$$\begin{aligned} E[X] &= n \sum_{j=1}^{n} \frac{1}{j} \\ &= nH_n \end{aligned}$$

We use two previous works in our analyses in Chapter 3; the uniform random approximation and singleton coupons.

```
Data: A list of coupons and one randomly selected coupon $c$. Coupons are
      selected uniform randomly
Result: A coupon $c$
begin
    $c \longleftarrow$ random($coupons$)
end
```

Algorithm 1: Uniform Random Coupon Collection

2.2.1 Harmonic Number Approximation

Recall that the censor collects n proxies in $E[X] = nH_n$ steps. There is no closed form expression for the harmonic number, H_n. Instead, the approximation with Euler Mascheroni Constant $\gamma \approx 0.5772156649$ and $H_n = \ln n + \gamma + 1/2n$ is used in this analysis [11].

2.2.2 Singleton Coupons

Variations of the CCP are explored in [22] where a single collector attempts to obtain a specific number of coupons out of n total coupons. As part of this analysis, we use the number of singleton coupons that Myers et al. provide. Singleton coupons are coupons that have been selected exactly once after all of the other coupons have been collected. Most coupons are duplicates and only a few appear only once. On average, the number of singleton coupons is equal to the harmonic number H_n.

2.3 Load Balancing

From the perspective of the proxy distributor, the proxy distribution problem is a form of load balancing where some clients are malicious and the proxy distributor would like to allocate as many proxies as possible to honest clients. Load balancing strategies are used to evenly allocate resources in distributed systems and have historically been modelled as balls and bins [19].

2.3.1 Maximum Load

The evenness of the distribution is formalized as the *maximum load*. The maximum load is equal to the number of balls in the bin that has, with high probability[1], the maximum number of balls over all the bins. The closer that the maximum load is to a perfect distribution of m balls over n bins, m/n, the more even the distribution, and thereby the load is more evenly balanced.

2.3.2 Uniform Random Distribution

The maximum load of a uniform random placement of balls in bins where $m = n$ is well known to be approximately $\log n/\log\log n$ with high probability [12]. See Lemma 5.1 below from the analysis in Chapter 5 of *Probability and Computing* [19].

Lemma 1 *When n balls are thrown independently and uniformly at random into n bins, the probability that the maximum load is more than* $3\ln n/\ln\ln n$ *is at most* $1/n$ *for n sufficiently large.*

2.3.3 Power of 2 Choice

The *power of 2 choice* algorithm for load balancing shows that giving two random choices instead of a uniform random placement results in an exponential improvement in the maximum load [20]. Figure 2 outlines the power of 2 choices algorithm that randomly selects two bins from the total list of bins. To determine the bin that is selected, the algorithm checks the load of each bin. The bin with the *least* number of client assignments is returned.

The maximum load in the uniform random case is $\frac{\log n}{\log\log n} + O(1)$ where the number of balls is equal to the number of bins [5]. In the two-choice algorithm, the fullest bin with high probability has $\frac{\log\log n}{\log 2} + O(1)$ balls where the number of bins selected randomly is 2. Increasing the number of bins ≥ 2 yields only a constant factor of improvement. The heavily loaded case, where the number of balls strictly increases over the number of bins, $m >> n$, was found to have a maximum load of

[1]with high probability means at least $1 - O(1/n)$

```
Data: A list of bins $bins$, two randomly selected bins $b_1$ and $b_2$ with total
      number balls, or loads, $l_1$ and $l_2$ respectively
Result: One of $\{b_1, b_2\}$
begin
    $b_1 \longleftarrow$ random($bins$)
    $b_2 \longleftarrow$ random($bins$)
    $l_1 \longleftarrow load(b_1)$
    $l_2 \longleftarrow load(b_2)$
    if $l_1 == l_2$ then
        return random($b_1, b_2$)
                                        /* break a tie */
    else if $l_1 < l_2$ then
        return $b_1$
                                        /* $b_1$ is least loaded */
    return $b_2$
                                        /* $b_2$ is least loaded */
end
```

Algorithm 2: Power of Two Choices

$m/n + O(\log\log n)$ [6].

2.4 Tor

One of the most widely used distributed, anonymous networks is the Tor onion network. Tor is useful for avoiding traffic analysis, a form of internet surveillance, but any anonymous network on its own is insufficient for censorship circumvention. For example, Tor packets are identifiable from telltale signs in the format of traffic sent to the Tor network. Additionally, these packets are easily blacklisted by censors because onion relay IP addresses in the onion network are publicly known. Tor includes various obfuscation protocols, known as `pluggable transports`, to hide distinguishing characteristics of packets from censors. These protocols are run by Tor `bridges`; bridges are single-hop proxies into the Tor onion network. Bridges are run by volunteers in a trusted proxy environment outside of the censor boundary. Tor needs to communicate the address of a bridge to legitimate users securely. This is another form of the proxy distribution problem; to distribute secret bridge addresses (proxies) only to legitimate users, not to censors.

A user requests bridge addresses from Tor by emailing Tor's bridge service. The bridge service, `bridgedb`, selects three bridge addresses at a time and rate limits these requests. For example, a request from a single email address can request bridges once daily. The `bridgedb` data structure is a hashring where an index into the keyspace is generated by hashing unique user information into a subring that is rotated on a scheduled interval.[2] This index gives the first bridge address, the other two bridge addresses are the successors of the first index in the ring. The censor can enumerate bridges one-by-one by posing as a legitimate user. Ling, Luo, Yu, Yang, and Fu showed in [16] that Tor bridges can be enumerated with bulk emails via the Hypertext Transfer Protocol Secure (HTTPS) protocol.

Coming up next, in Chapter 3, we'll analyze the expected time to collect all of the proxies using the needle algorithm in the context of the coupon collector problem. We'll also refer to the maximum load in the load balancing analysis. Later on, in Chapter 4, we'll compare the uniform random, power of 2 choices, and Tor's distribution mechanism with the needle algorithm in our evaluation.

[2]https://gitweb.torproject.org/bridgedb.git/tree/bridgedb/distributors/email/distributor.py

Chapter 3

Needle Algorithm

We present a new algorithm, the *needle* algorithm, and compare it to two well-known algorithms, uniform random and power of 2 choices, as well as Tor's bridge distribution scheme, in order to contrast their respective trade-offs and suitability under differing system goals. The primary goal of the needle algorithm is to preserve some proxies from within the larger list of n proxies. Another goal is to maintain practical load balancing for the majority of the proxies. It achieves these goals by breaking the list of proxies into sublists. It moves through the proxies jumping from sublist to sublist, referred to as `GIANTSTEPS` or g. These steps isolate some proxies from distribution.[1] The algorithm has the following parameters:

- *proxies*: a sorted list of proxies ordered by load
- n: the total number of proxies
- g: the number of steps around the proxy ring
- s: the number of proxies in each g step
- *move*: indicates a move to the next step
- px_1, px_2: two randomly selected proxies from within the *sublist* of the current step g, sampled with replacement

[1]Those readers familiar with the algorithmic composition techniques of jazz chord progression may draw a parallel from the restriction of proxies to the restriction of notes in a key change. These are reminiscent of Coltrane's Giant Steps [15].

```
**Data:** $proxies$, $n$, $s = \lfloor n/g \rfloor$
**Result:** One of $\{px_1, px_2\}$
begin
    $window \longleftarrow (start, end)$
    if $move > 1$ then
        /* a giant step around the proxy ring */
        $start \longleftarrow (start + s)\ \%\ n$
        $end \longleftarrow (end + s)\ \%\ n$
        $move \longleftarrow 0$
    $move \longleftarrow move + 1$
    $sublist \longleftarrow (proxies\ [start, end])$
    $px_1 \longleftarrow \text{random}(sublist)$
    $px_2 \longleftarrow \text{random}(sublist)$
    $l_1 \longleftarrow load(px_1)$
    $l_2 \longleftarrow load(px_2)$
    if $l_1 == l_2$ then
        return $px_1$
                                        /* break a tie */
    else if $l_1 > l_2$ then
        return $px_1$
                                        /* $px_1$ is heaviest loaded */
    return $px_2$
                                        /* $px_2$ is heaviest loaded */
end
```

Algorithm 3: Needle Algorithm

The algorithm selects two proxies uniform randomly in each giant step g. Using reverse power of 2 choices, it then chooses the *heavier loaded* proxy and returns this proxy to the collector. (Note that the standard power of d choices algorithm chooses the lighter loaded proxy.) The step begins at the bottom of the list of proxies and is moved step-wise around the list, eventually wrapping around, so that all proxies have an opportunity to be selected, shown in Figure 3.1.

Some number of proxies are less likely to be selected as step g moves. For example, the proxies in previous selections left behind are referred to as *needles* because they are hidden from the current selection. Additionally, proxies that have a lower chance of appearing in the random selection, e.g., the last few distinct proxies to be selected, have a lower probability of appearing in any selection. As

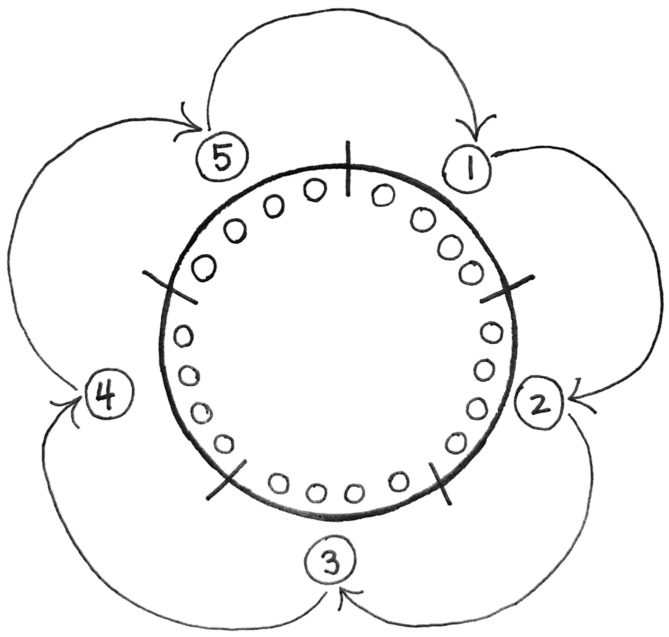

Figure 3.1: 5 giant steps around a ring of 20 proxies

g moves, it also unbalances the load on proxies thereby increasing the number of selections that the censor must perform because there are more duplicate selections (misses).

3.1 Coupon Collector Ring

One may find a parallel between the Coupon Collector Problem CCP and the problem of a censor who collects, not coupons, but proxies. Consider both the coupon and proxy as objects in some vast space, where all of these objects have the same chance of being selected. Each object is replaced after it is selected so that any object may be selected multiple times. It is increasingly difficult to collect a distinct object because most objects are selected more than once. The rarest object is the one that occurs at the very end to complete the collection. The selection of this very last unique object ends the game, since only the selection of distinct objects counts towards progress in this problem.

We can frame the needle algorithm as a CCP variant, we call this the Coupon Collector Problem Ring (CCP-R) variant. Suppose that the coupon collector organizes collections from several different neighbourhoods. Each neighbourhood distributes a different set of coupons, and each of the coupons in each set is distinct. The coupon collector needs to travel through all of the neighbourhoods (these are conveniently arranged in a ring). The collector can't magically move from place to place and does not zigzag from neighbourhood to neighbourhood. If the collector misses a coupon from the first neighbourhood, then he must wait until all of the other neighbourhoods are visited before returning to collect the missed coupon. The missed coupon is rarer than all of the previously collected coupons. It is also rarer than all future coupons up until the point at which the collector returns to the specific neighbourhood with the missed coupon.

In our CCP-R variant, the reverse power of 2 choices gives the collector a decision to make; he selects 2 coupons at random. If he already has more of 1 coupon than the other, he must keep the coupon that is less rare, and return the rarer coupon back to the store in the current neighbourhood.

There is an expected number of attempts $E[X]$ by the collector to collect n distinct objects. This can be extended to a censor, whose problem it is to *enumerate*, or discover, all the proxies. We use $E[X]$ to analyze the amount of effort that it would take a censor, on average, to enumerate all proxies. To simplify this analysis, we do not consider the ratio of honest to malicious clients and, for now, assume there is one malicious client, the censor. Honest and malicious client rates are examined in Section 3.4, bystanders.

The needle algorithm is partly randomized and partly deterministic. The uniform random selection of 2 proxies is randomized, and the reverse power of 2 choices is deterministic. In addition, the movement of the steps is also deterministic. Because the steps move clock-wise around the ring, proxies are enumerated randomly within deterministic batches. Moving the steps around the ring serves two functions; 1) by limiting the size of proxies in a step, it squeezes the number of possible proxies that a censor can enumerate in each assignment, and 2) some proxies are left behind. This makes it more difficult for the censor to enumerate a complete collection of proxies due to these rare proxies (not unlike needles in a haystack). Restricting the size of the sublists in a step tends towards a deterministic enumeration of all of the proxies, so one must take care to not set too many giant steps.

In Figure 3.2, we see how different step sizes affect the enumeration of proxies. With one-step, the enumeration time is the slowest, because the needle proxies must be discovered within the largest pool of size n. Enumeration time starts to decrease with 2-steps, because the needle proxies are more likely to be selected with a smaller pool size of $n/2$. The enumeration time decreases with each increase in giant step g. The algorithm only functions as expected up to $g = n/2$ where there are at least 2 proxies for each step. If there is only a single proxy in each step, as shown in the bottom right corner, the proxies are enumerated one after the other in n assignments, because our reverse power of 2 choices algorithm cannot function as intended. We therefore restrict the giant step size to $1 \geq g \leq \frac{n}{2}$ so that the proxies are never enumerated in deterministic order.

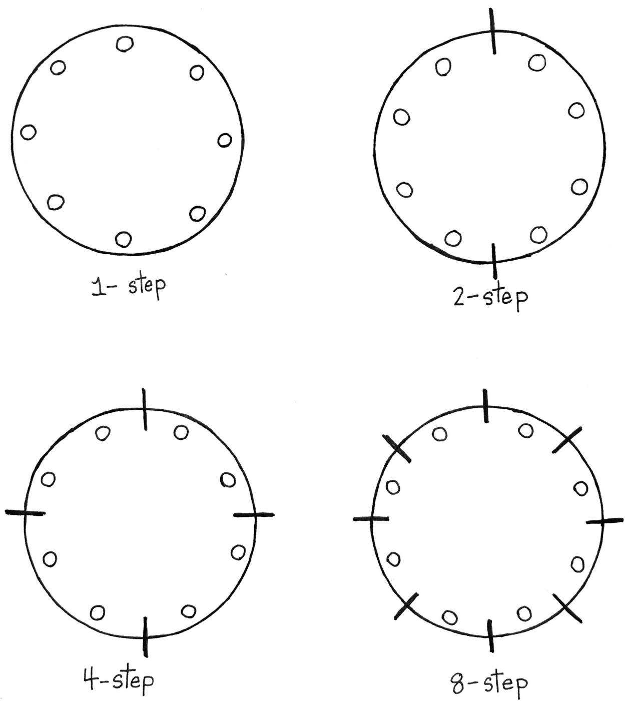

Figure 3.2: Rings of 8 proxies each with steps g=1,2,4,8

3.2 Enumeration Analysis

Recall that the enumeration of our collection of proxies occurs when the censor learns all of the proxies through assignment. The enumeration of proxies is modeled as the complete collection of coupons. The CCP-R variant is used to calculate the average time to collect all distinct proxies.

Parameter Name	Description
n	total number of proxies
g	number of steps around a proxy ring
s	size of proxies allocated in each step n/g
H_n	harmonic number
px_i	the i^{th} proxy
p_i	probability that the i^{th} proxy is enumerated
p	sum probability that proxies 1 to n are enumerated
X_i	random variable of assignments to enumerate the i^{th} proxy
X	random variable of sum of X_i from 1 to n
$E[X]$	expected time on average to enumerate all proxies

Table 3.1: Notation used in the Enumeration Analysis

There are two different, yet intertwined, flavours of probabilities in our algorithm; the probability of *selection* and the probability of *choice*. The probability of selection is uniform random and is restricted by the size of the giant step g. When a pair of proxies is selected uniform randomly from the total collection of proxies, this is referred to as the selection. When we refer to likelihood of selection, we are talking about the probability of selection in relation to the coupon collector problem. Because we sample proxies randomly with replacement, a proxy can be selected twice in a pair. The probability of choice relates to the order in which a proxy is selected as a pair. The likelihood of choice is dependent on the load of a proxy as it affects the choice in the reverse power of 2 choices algorithm. Probability of choice is important in our analyses, particularly in the load balancing analysis.

3.2.1 Algorithm Termination

We define two lemmas below that we will use to approximate upper and lower bounds on the average enumeration time. We define the algorithm termination in terms of the rarest proxy, the last distinct proxy that is selected, px_n. Proxy px_n has the probability of selection = $1/n$ making it the rarest proxy out of all of the other proxies. Note that this proxy must have load = 0 because it was never previously chosen out of a selected pair. Recall that the reverse power of 2 algorithm chooses the proxy with the highest load out of a pair of proxies. The rarest proxy can only be chosen if it is compared against itself. The only case where this occurs (when the proxy appears with probability of selection $1/n$) is when this proxy is selected in a pair.

Lemma 2 *A pair of the rarest proxy terminates the algorithm.*

Assume we have the rarest proxy px_n that has probability equal to $1/n$ of being selected in the reverse power of 2 choices algorithm. Suppose that this proxy is selected twice in a row and forms a pair. If the choice from this selected pair does not terminate the algorithm, there must be some other proxy in the collection of n proxies that has not yet been chosen. If this statement is true, then proxy px_n must have a probability greater than $1/n$ and our initial assumption is false, leading us to a contradiction. Therefore, the proxy with probability $1/n$ terminates the algorithm if it is selected twice in a row, as a pair. In other words, the least likely, rarest proxy must occur as a pair for the algorithm to terminate.

Lemma 3 *The probability of enumerating the rarest proxy is at least $1/n^2$.*

By Lemma 2, the needle algorithm terminates after the rarest proxy is selected in a pair. The probability that the rarest proxy is selected is $1/n$. We sample with replacement, so the probability that the rarest proxy is selected twice in a pair, by counting argument, is $p = (\frac{1}{n})(\frac{1}{n}) = 1/n^2$.

3.2.2 Upper Bound

We use the number of singletons described in Chapter 2, H_n, directly to provide an upper bound on the average number of pairs selected, $E[X]$, before all of the proxies are enumerated. Singletons are proxies that occur only once and so have the lowest probability of selection. These proxies need to be paired with other proxies that have lighter loads to ensure that they are chosen by the reversed power of 2 choice algorithm. (They also need to appear as the first item in the pair for the tie-breaking logic.) Most of the singleton proxies occur at the end of the selection process, so the probability of selection for each of them is closer to $1/n$ than n/n. In the worst case (for the censor), all of the singleton proxies occur at the end of the collection. Since most of the proxies have load > 1 as n increases, there is little chance that singleton proxies are chosen against non-singleton proxies, because most of these non-singleton proxies have appeared numerous times already and have higher assignment loads.

Lemma 4 *The probability of collecting all of the singleton proxies is*

$$\sum_{i=1}^{\lceil H_n \rceil} \left(\frac{i}{n^2}\right)$$

The probability of collecting all of the singleton proxies is bounded above by the likelihood that the singleton proxies are selected in a pair with the rarest proxy px_n, the worst case for a censor. The rarest proxy px_n is selected with probability $\frac{1}{n}$. The probability that some i^{th} singleton proxy is selected in a pairing with px_n is $p_i = (\frac{i}{n})(\frac{1}{n})$.

If we pair all of the singleton proxies with the rarest singleton proxy, px_n, we get the probability pairs $(\frac{1}{n})(\frac{1}{n}) + (\frac{2}{n})(\frac{1}{n}) + ... + (\frac{H_n}{n})(\frac{1}{n})$. There is no guarantee that H_n is an integer, indeed it will not be. Fractional pairings can't happen in our pairings, e.g. there is no half of a proxy selected. Because we are dealing with an upper bound, we say that the number of singleton proxies is $\leq \lceil H_n \rceil$. In addition, we need an even number of pairings, so $\lceil H_n \rceil$ is rounded to an even number. Generally stated, the probability of this pairing over all H_n singletons is:

$$\begin{aligned} p &\leq \sum_{i=1}^{\lceil H_n \rceil} \left(\frac{i}{n}\right)\left(\frac{1}{n}\right) \\ &= \sum_{i=1}^{\lceil H_n \rceil} \left(\frac{i}{n^2}\right) \end{aligned}$$

Now, we are ready to present the upper bound on the average number of assignments before all of the proxies are enumerated. Recall that X_i is a random variable representing the number of assignments needed to enumerate the i^{th} proxy, therefore $E[X_i]$ is the average number of assignments to collect the i^{th} proxy. $E[X]$ is the average number of assignments needed to enumerate all of the proxies. We denote the i^{th} proxy as px_i where the last selected, rarest proxy is px_n with probability $1/n$. The second rarest proxy is px_{n-1} with probability $2/n$. The first proxy selected is px_1 with probability 1.

Theorem 5 *Upper bound on* $E[X] \leq \frac{n^2 H_{\lceil H_n \rceil}}{g}$

Proof. We use the probability of collecting all of the H_n singleton proxies from Lemma 4 to provide an upper bound on the average number of assignments needed to enumerate all of the proxies, $E[X]$.

Like the classic CCP, this follows a geometric distribution, so $E[X_i] = \frac{1}{p_i}$.

$$E[X_i] = 1/p_i = \frac{1}{\frac{i}{n^2}} = \frac{n^2}{i}$$

The expected number of assignments to find all n proxies is bounded above by the sum of $E[X_1]$ to $E[X_n]$, where $n = \lceil H_n \rceil$:

$$E[X] \leq \sum_{i=1}^{\lceil H_n \rceil} \left(\frac{n^2}{i} \right)$$

$$E[X] = n^2 \sum_{i=1}^{\lceil H_n \rceil} \left(\frac{1}{i} \right)$$

$$E[X] = \left(n^2 \right) \left(H_{\lceil H_n \rceil} \right)$$

When giant step g is very small, the algorithm churns on a few proxies, meaning that a small proportion of proxies have high loads. By dividing n proxies into $g > 1$ giant steps, where each step has a sublist of size s, we make the algorithm more load balanced. In other words, the distribution of proxies in each step is smaller so the censor gets more chances to find the singletons. We rework the upper bound to incorporate the number of giant steps g.

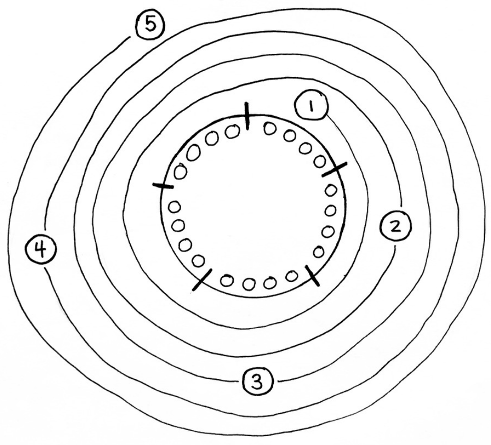

Figure 3.3: Worst case enumeration in 5 giant steps.

We have g giant steps and H_n singleton proxies; g divides the enumeration time into smaller sized sublists, making it proportionally easier for the censor to enumerate all n proxies. In the worst case (for the censor), all of the singleton proxies are located in different steps, so all of the steps wait for the other steps' singletons to be enumerated before the algorithm can terminate. Figure 3.3 shows a collection of proxies with 5 giant steps. The first enumerated sublist occurs in the top right portion of the ring (denoted (1)). The algorithm proceeds clock-wise, moving through each sublist one by one. We divide the upper bound by the number of steps g to obtain:

$$E[X] \leq \frac{n^2 H_{\lceil H_n \rceil}}{g}$$

3.2.3 Lower Bound

Before we present the lower bound on the average number of assignments before all of the proxies are enumerated, we consider the probability of singleton proxies that are very likely to be chosen. The reverse power of 2 choices relies on the load of the proxy in order to choose a proxy from a selected pair. Consider a singleton proxy that by definition has a lighter load because it is selected infrequently. When this proxy is paired with some other proxy with a higher probability of selection, the singleton proxy has *less* chance to be chosen, because the proxy with the higher probability of selection has a higher chance of having more load. In other words, this singleton proxy has both a lower probability of selection and lower probability of choice.

We concern ourselves with a lower bound, so we would like to know the type of scenario in which a singleton proxy has the higher probability of choice, meaning that the censor is able to enumerate proxies more easily. A singleton proxy paired with itself is guaranteed to be chosen and thus has the highest probability of choice out of any other pairing. We would like to know the probability of a pair of singleton proxies to appear.

Lemma 6 *Singleton proxies occur in a pair with probability* $p \geq \sum_{i=1}^{\lfloor H_n \rfloor} (\frac{i}{n})^2$

A repeated pair selection of the singleton proxies gives us the probability pairs $p = (\frac{1}{n})(\frac{1}{n}) + (\frac{2}{n})(\frac{2}{n}) + ... + (\frac{H_n}{n})(\frac{H_n}{n})$*. The probability of the* i^{th} *singleton proxy to be chosen is* $(\frac{i}{n})^2$*. The probability of this pairing for all* H_n *singletons is:*

$$p \geq \sum_{i=1}^{\lfloor H_n \rfloor} \left(\frac{i}{n}\right)^2$$

We use Lemma 6 in the following proof to obtain the lower bound on $E[X]$. The fastest enumeration of all of the proxies is to collect all of the singleton proxies the first time that they show up in a randomly selected pair, that is, their very first selection. We again use H_n singleton proxies, this time to bound the average number of assignments from below. We consider cases where $g \geq 1$ and we change the variable n to s because we draw from g sublists of size s.

Returning to the idea of a ring of proxies, if all of the sublists are enumerated one directly after the other, none of the other steps are "held up" by any of the

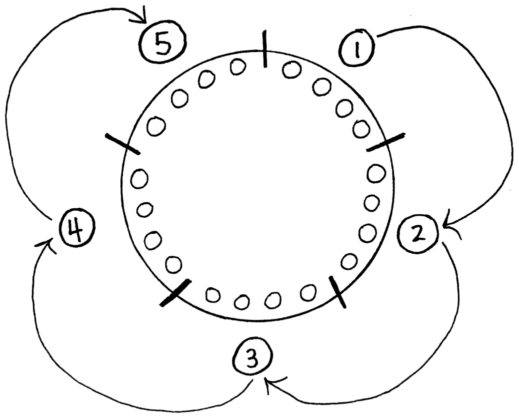

Figure 3.4: Best case enumeration in 5 giant steps.

other previous steps. This means that there are no extra revolutions necessary to enumerate the steps, as shown in Figure 3.4.

Theorem 7 *Lower bound on* $E[X] \geq (g)(s^2)H^{(2)}_{\lfloor H_n \rfloor}$

Proof. Using the probability of pairs from Lemma 6, we have a geometric distribution, so we can write:

$$E[X_i] = \frac{1}{p_i} = \frac{1}{(\frac{i}{s})^2} = \left(\frac{s}{i}\right)^2$$

The expected number of assignments to find all n proxies is bounded below by the sum of $E[X_1]$ to $E[X_n]$, where n is the number of singleton proxies, $n = \lfloor H_n \rfloor$:

$$\begin{aligned} E[X] &\geq \sum_{i=1}^{\lfloor H_n \rfloor} \left(\frac{s}{i}\right)^2 \\ &= s^2 \sum_{i=1}^{\lfloor H_n \rfloor} \frac{1}{i^2} \\ &= s^2 H^{(2)}_{\lfloor H_n \rfloor} \end{aligned}$$

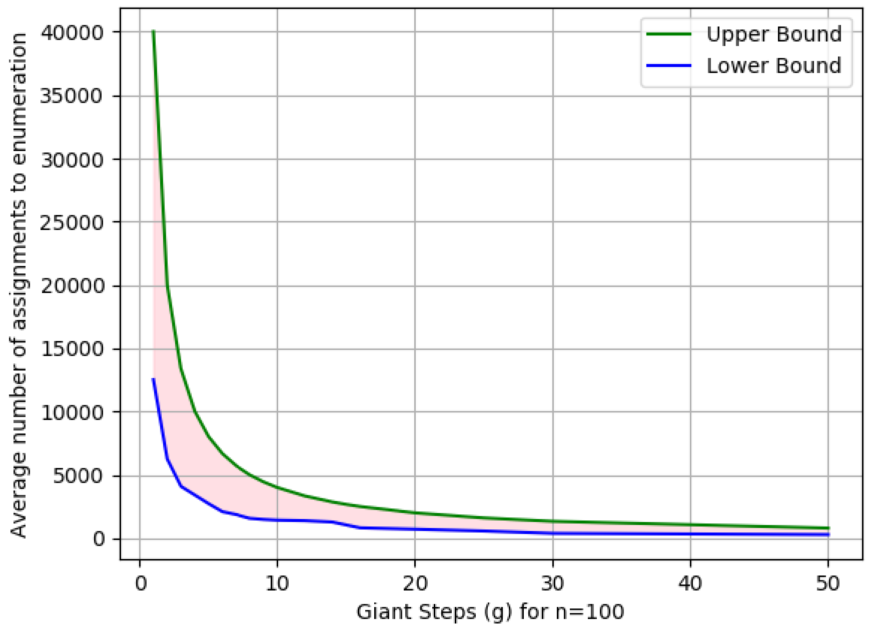

Figure 3.5: Upper and Lower Bounds for the Needle algorithm, where $n = 100$, $g = 1...50$

We have g such enumerations of sublists with size s, one per each g steps, therefore for all n proxies are bounded from below by:

$$E[X] \geq (g)(s^2)H^{(2)}_{\lfloor H_n \rfloor}$$

In Figure 3.5, we show the upper and lower bounds for 100 proxies with giant steps 1 to 50. The enumeration time drops for each giant step, particularly dramatically where the giant step is increased from 1 to 2. It levels out as the giant step increases, as these increases create an increasingly deterministic enumeration.

This concludes our discussion of bounds on the average number of assignments to enumerate all of the proxies in the needle algorithm. In the next section, we dive into a load balancing analysis to tell us more about how clients are assigned to proxies.

3.3 Load Balancing Analysis

We discussed maximum load as a metric for load balancing in Chapter 2. We cannot guarantee that the number of bins (proxies) is equal to the number of balls (assignments), so the maximum load does not tell the whole picture. We need to look at the least loaded and heaviest loaded extremes and find an average measurement and compare this to the *optimal load*. The optimal load is the load of each proxy if the system was perfectly balanced, that is, if the number of assignments were divided evenly over the number of proxies.

Parameter Name	Description
n	total number of proxies
m	total number of assignments in a series of trials
px_i	the i^{th} proxy
p_i	probability that the i^{th} proxy is enumerated
ℓ	optimal load
u	maximum load
$E[X]$	expected time on average to enumerate all proxies
d_1	distance between px_1 and $px_{\frac{n}{2}}$
d_2	distance between $px_{\frac{n}{2}}$ and px_n

Table 3.2: Notation used in the Load Balancing Analysis

Lemma 8 *Proxy loads increase linearly.*

We sort the proxies by their load, or their respective numbers of assignments, averaged over multiple trials. (Note that these are not static proxy numbers, but proxies that are numbered by their load.) We see that the loads have a linear relationship. This is because the probability of selection of the px_i proxy decreases linearly from $i = 1...n$. The probability of selecting the first proxy px_1 is $p_1 = \frac{n}{n} = 1$, the probability of the second px_2 is $p_2 = \frac{n-1}{n}$, $p_{n-1} = 2/n$, $p_n = 1/n$, increasing *linearly. This linear relationship between the probability of proxy selection is further reinforced by the probability of choice, where the maximum loaded proxy is always chosen over a lower load.*

Lemma 9 *The middle proxy, $px_{n/2}$, has the optimal load ℓ.*

Let the number of total assignments in a series of trials be m, and $E[X]$ be the average number of assignments before all proxies are enumerated where $m >> E[X]$. The optimal load is $\ell = m/n$. Proxy $px_{n/2}$ has the probability of selection equal to $p_{\frac{n}{2}} = \frac{n/2}{n}$ in the reverse power of 2 choice selection. It is $n/2$ more likely to be chosen from the selection than proxies px_1 to $px_{\frac{n}{2}-1}$ and $n/2$ less likely to be chosen over proxies $px_{\frac{n}{2}+1}$. Its placement at the midway point of choice means that it is also at the midway point of the load ℓ.

Theorem 10 *The maximum load, u, of proxy px_1 is twice as big as the optimal load, 2ℓ, if all of the proxies are enumerated.*

We know by CCP that the minimum load is at least 1 if all of the proxies are enumerated. By Lemma 9, we see that the middle proxy $px_{\frac{n}{2}}$ has the optimal load. Since the proxy load has a linear relationship by Lemma 8, the distance, d_1 between px_n and $px_{\frac{n}{2}}$ is equivalent to the distance, d_2 between $px_{\frac{n}{2}}$ and px_1. $px_{\frac{n}{2}}$ has the optimal load and the function is increasing linearly, therefore px_1 has twice the optimal load $u = 2\ell = (2)(\frac{m}{n})$.

3.4 Bystander Analysis

We integrate honest users into our analysis where previously we only included the censor's problem of enumerating all of the proxies. We want to know the likelihood that a proxy has been assigned to an attacker, *proxy exposure*, and the proportion of honest users, or *bystanders*, that are affected by proxy exposure. We draw inferences from the load of each proxy and the probability that a client is an attacker to evaluate bystanders in section 4.4.

We define an *exposed* proxy as a proxy that has at least one insider attacker assigned at any point in time. In a single assignment of a client to a proxy, the probability that the client is an attacker is given as p_a. The total number of assignments in a proxy is equivalent to its historical load h.

Parameter Name	Description
p_a	probability that a client is an attacker
p_x	probability that a proxy is exposed
p_{nx}	probability that a proxy is not exposed
p_b	probability of bystander clients
h	historical load on a single proxy
ℓ	optimal load
u	maximum load

Table 3.3: Notation used in the Bystander Analysis

Lemma 11 *The probability that a proxy is not exposed is* $(1 - p_a)^h$.

The number of client assignments that a proxy received is equal to the historical load h. The probability that all h of these assignments are attackers is an "and" relationship. We multiply the probabilities of attackers per each assignment, h times. Therefore, the probability that all of the assigned clients on a proxy are malicious is $(p_a)^h$.

For a proxy to not be exposed, then it must never have had any attackers assigned to it. The probability that none of the clients assigned are attackers is $(1 - p_a)^h$. *Then the probability that a proxy is not exposed is the same as the probability that none of the clients are attackers:*

$$p_{nx} = (1 - p_a)^h$$

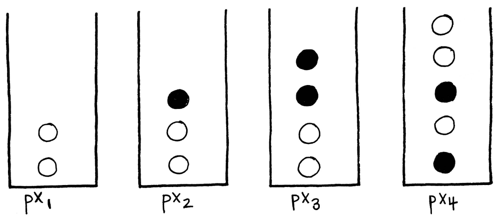

Figure 3.6: Proxies px_1 to px_4 with honest and attacker clients.

We generalize the probability of exposure to apply to any proxy in the system using a bound on the maximum load. This is interesting for our bystander analysis because we observe that the probability that a proxy is exposed more closely approaches 100% as the maximum load increases, thereby creating more bystanders.

Lemma 12 *The probability that any proxy is exposed is $p_x \leq 1-(1-p_a)^u$*

The height of any proxy must be less than or equal to the maximum height, u, by definition. Without knowing the load on any individual proxy, we can say that the likelihood that an arbitrarily selected proxy is not exposed is less than or equal to the likelihood that the maximum loaded proxy is not exposed.

$$p_{nx} = (1-p_a)^h \leq (1-p_a)^u$$

.

Conversely, the probability that any randomly selected proxy is exposed is less than or equal to the probability of exposure on the proxy that has the maximum load.

$$\begin{aligned} p_x &= 1 - p_{nx} \\ &\leq 1-(1-p_a)^u \end{aligned}$$

The proxy that has the highest load has the highest chance of exposure. In Figure 3.6, we give an example with four proxies, px_1 to px_4 that have attacker clients represented as filled circles, and honest clients shown as empty circles. There are no bystanders in proxy px_1 because it is not exposed. Proxies px_2 and px_3 both have two bystanders because they each have two honest clients and the proxies are exposed. There are three bystanders on proxy px_4. We've shown how the maximum load affects the probability of assigned attackers in a proxy, now we will compare the relationship of load balancing to bystanders.

Theorem 13 *The probability of bystanders on a proxy is*

$$p_b \leq (u)(1-p_a)(1-(1-p_a)^u)$$

Proof. Honest clients occur with probability $1-p_a$ on any proxy. For an honest client to be considered a bystander, there must be at least one attacker assigned to the proxy. We consider the other attackers on a proxy as wasted resources on the part of the attacker.

We know from Lemma 12 that the probability that any proxy is exposed is $p_x \leq 1-(1-p_a)^u$. If $p_x = 0$ then there are no bystanders, because the proxy is not exposed. (In order for honest clients to be defined as bystanders, the proxy must be exposed by at least one attacker.) Therefore, we can say that the probability of bystanders on any proxy is less than the probability of bystanders on the proxy with the maximum load.

$$p_b \leq (u)(1-p_a)(1-(1-p_a)^u)$$

We have shown that the amount of collateral damage, expressed by the probability of bystanders on a proxy, relies on the proxy load and the probability of attackers in each assignment. There are other interesting stories to tell about the impact of unbalancing proxy loads for proxy preservation. If we have a system that is perfectly balanced, with all proxy loads equal to ℓ, intuitively we suspect that there will be more innocent bystanders b. We may reason that with lower proxy loads, such as those held in the needle algorithm of proxies px_1 to $px_{\frac{n}{2}-1}$, result in fewer bystanders because the likelihood of a malicious client attacker is reduced. We may also reason that in higher loaded proxies, there is more chance of proxy enumeration, but more wasted resources by the attacker. These musings are examined in Chapter 4 through empirical data based on simulations of systems using the needle algorithm, uniform random distribution, power of 2 choices, and Tor's `bridgedb` proxy distribution strategy.

Chapter 4

Evaluation

4.1 Simulation Model

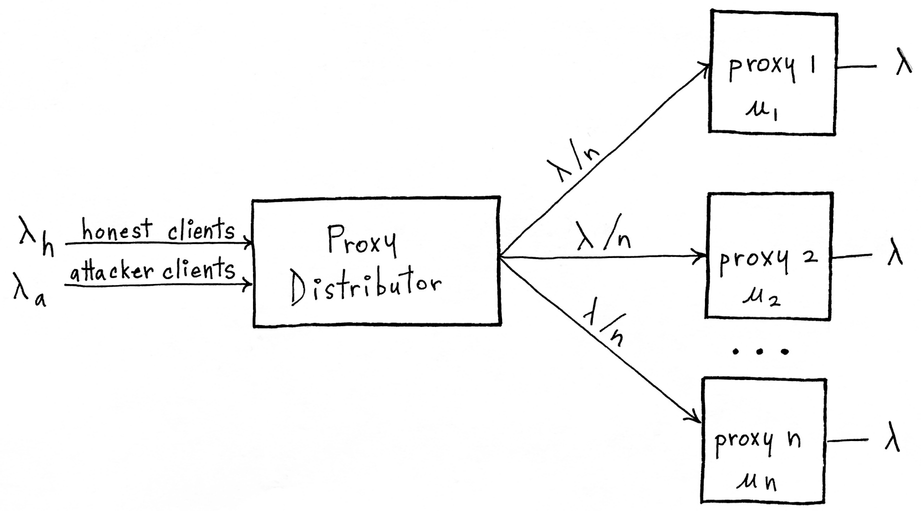

Figure 4.1: M/M/c/k with uniform random distribution

Before considering details of the simulation experiments, we formalize the simulation model and how it applies to the needle algorithm proxy distribution strategy. The notion of static and dynamic models stems from static and dynamic routing in networks. Mitzenmacher redefined static and dynamic models for task allocation and applied this to the balls and bins model [20]. In this context, a ball is a task and a bin is a processor. The primary distinguishing features of the static model is that there are a finite number of tasks that are assigned to processors and these tasks do not leave. The static system model completes its execution when all of the tasks are allocated. This model execution is generally represented in a bipartite graph. A static model applied to the proxy distribution problem assigns a finite number of clients to proxies in one round.

The dynamic model captures scenarios that are more realistic than what may be represented in the static model. For example, tasks in the dynamic model may enter and leave the system over time. In the open dynamic model, there may not be a fixed number of tasks. The closed dynamic model allows for tasks to enter and leave over time, but there is a fixed number of arrivals. A dynamic system does not have a final termination time as in the static model. The advantage of a dynamic model is that one can observe the behaviour of a system over time.

We are interested in how proxies are enumerated by a censor throughout the system's lifetime, so the open dynamic model suits our scenario well. Utilizing an open dynamic model allows us to observe trends and compare different approaches temporally. However, our clients don't *leave* in the traditional sense; we assume that once a proxy is discovered, the knowledge of a proxy cannot leave or exit.

Parameter Name	Description
λ_h client arrival rate	intensity of client arrival rate $1/\lambda$
λ_a attacker arrival rate	intensity of attacker client arrival rate $1/\lambda$
n	number of proxies

Table 4.1: Variables in the Simulation

In addition to the dynamic model, the queue model is a standard that suits the problem of proxy distribution well. Mitzenmacher models the power of two choice algorithm as an idealized process, or a queue system of infinite size, that is later related to a finite system by bounding the error between the two [20].

Our simulation is best described as a feed-forward network of M/M/c/k queues. This is similar to a load balancing system with a constant number of servers. However, in this scenario, the load balancer is in fact the proxy distributor that is responsible for distributing clients to proxies. Figure 4.1 shows the network queue layout where the proxy distributor assigns clients to proxies uniform randomly, where the arrival of clients to proxies is evenly distributed λ/n.

Clients arrive with a Poisson arrival distribution with variable λ clients per minute. This means that, on average, one client appears at every $1/\lambda$ minutes and we define this as the arrival intensity. In this model, we are additionally concerned with honest and malicious client arrival rates. As the proportion of attackers in the system increases, the arrival rate, or intensity of malicious client arrivals increases.

Each proxy has a service time that is exponentially distributed. This is not a traditional service rate because the system assumes short-lived client connections. The important function of the server is to keep a record of each client assignment that can be used by the censor to discover clients in the system. In other words, even if the client leaves the system, the censor still gains knowledge of the proxy and may impact the client's future connections to the known proxy, effectively mounting a potential future blocking attack on the proxy.

Parameters. Parameters in the simulation control the rate of honest and malicious client arrival intensity, the total number of proxies, and the size of the step for the needle algorithm. These are sweeping parameters; the simulation runs across a range of values for each parameter to produce a variety of conditions for the analysis. The analysis operates on the dependent variables such as maximum load and the expected time to overtake the system. The variable names and descriptions are outlined in table 4.1.

The simulator was written in Python using simpy, a discrete event based simulator.[1] The evaluation uses numpy[2] and matplotlib[3] for data manipulation and graphing.

4.2 Enumeration Results

Giant step analysis. We ran the needle algorithm in the simulator, with only malicious attacker clients, to observe how many assignments a censor needs to enumerate all of the proxies. In section 3.2.2, we analyzed the giant step version of

[1]https://simpy.readthedocs.io/en/latest/
[2]http://www.numpy.org/
[3]https://matplotlib.org/

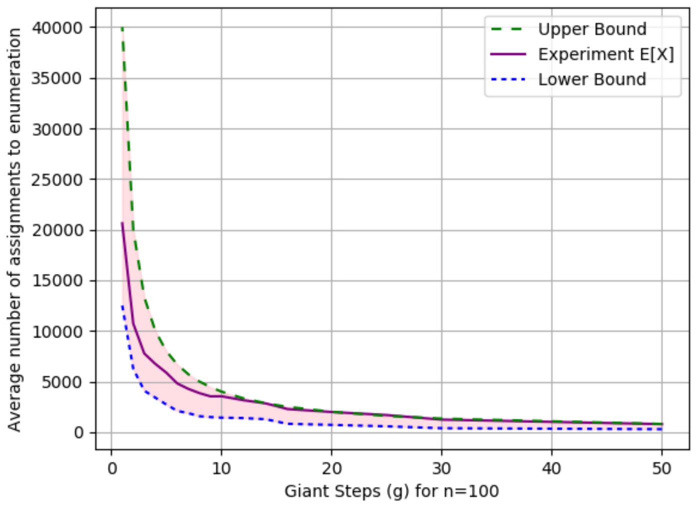

Figure 4.2: Giant step enumeration comparison for 100 proxies

the needle algorithm where step size $g > 1$. Results are shown in Figure 4.2 for 1000 trials using 100 proxies in each experiment. The experimental results are shown in the black, solid line. The upper and lower bounds, $E[X] \leq \frac{(n^2)H_{\lceil H_n \rceil}}{g}$ and $E[X] \geq (g)(s^2 H^{(2)}_{\lfloor H_n \rfloor})$ respectively, are shown in dashed lines.

Table 4.2 shows the numerical data from these 1000 trials of varying step size for 100 proxies, and the calculated upper and lower bounds.

Enumeration Comparison. We compare the four different algorithms in Figure 4.3. The y-axis shows the average enumeration time over 1000 trials where a higher number of assignments is favourable because it takes longer for a censor to discover all of the proxies. The x-axis shows the trials over different sizes of n proxies. We know from the Coupon Collector Problem CCP that the uniform random distribution of coupons results in nH_n total assignments before collection. We see that the uniform random distribution of Tor's `bridgedb` follows a similar enumeration. We use the regular power of 2 choices algorithm to show how a more optimal load balancing algorithm results in faster enumeration [27]. In the upcoming section, we'll take a closer look at the load balancing properties of each of these algorithms.

Sublist (s)	Step (g)	Upper Bound	Experiment	Lower Bound
100	1	40000	**20619**	12500
50	2	20000	**10718**	6250
33	3	13333	**7763**	4083
25	4	10000	**6738**	3403
20	5	8000	**5869**	2722
16	6	6667	**4800**	2091
12	8	5000	**3850**	1568
10	10	4000	**3537**	1424
8	14	2857	**2839**	1276
5	20	2000	**1987**	712
4	25	1600	**1683**	569
2	50	800	**788**	285
1	100	400	**100**	100

Table 4.2: Experimental results for 100 proxies over 1000 trials

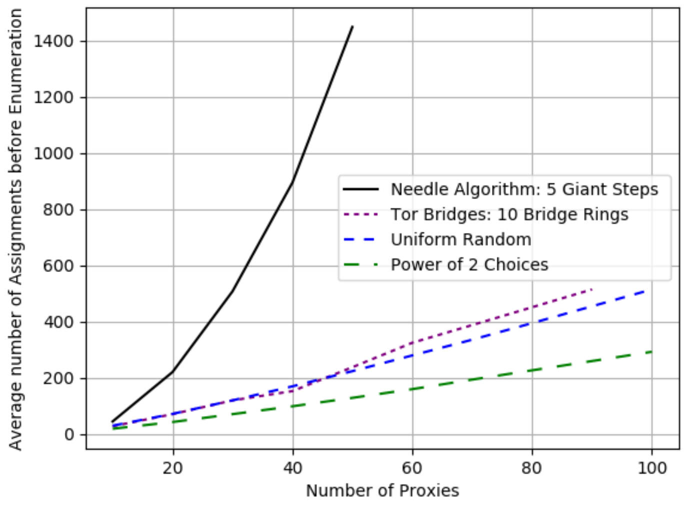

Figure 4.3: Number of assignments before enumeration for each algorithm for n=10 to 100 proxies.

4.3 Load Balancing Results

Recall that in the analysis, Lemma 9 in section 3.3 stated that $px_{n/2}$ has the optimal load. In the experiments shown in Figure 4.4, we ran the needle trials until all of the proxies were enumerated and then kept running the trials until twice the number of enumeration time. We see that the load balancing is divided into two halves centred around the optimal load. We also note that the maximum load for each of the experiments is no more than 4% of the total load.

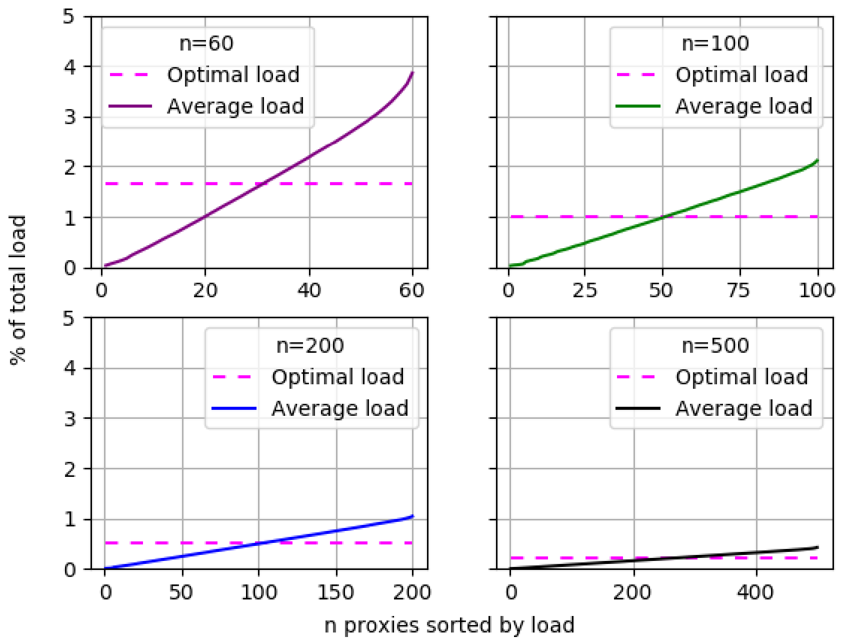

Figure 4.4: Needle Load Balancing until all proxies are enumerated twice for $n = 60, 100, 200, 500$.

We wish to observe and compare the load balancing tendencies of each of the four algorithms. The x-axis in Figure 4.5 orders all of the proxies by their respective loads. The y-axis shows the percentage of the total load that each proxy holds. We ran each algorithm for 5000 assignments with a total of 100 proxies.

A distinguishing feature of each algorithm is their varying degree of even distribution of assignments to proxies. The power of 2 choices algorithm gives the most evenly balanced distribution. The uniform random distribution results in the next best load balancing. Tor's mechanism is less balanced than uniform random but more balanced than the needle algorithm. The needle algorithm results in the worst load balancing.

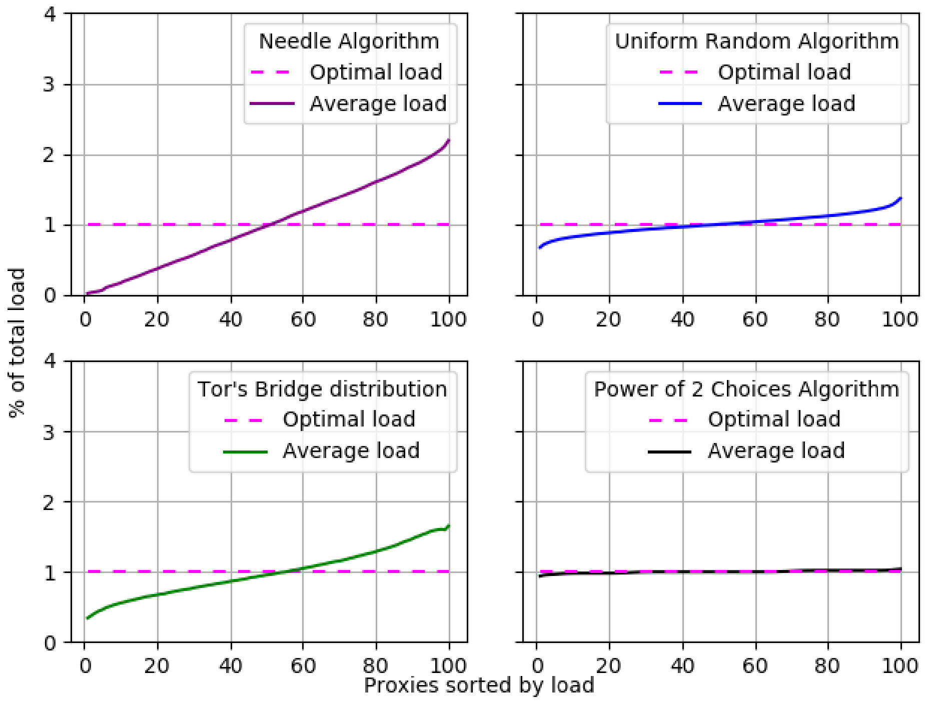

Figure 4.5: Load balancing results for each algorithm for n=10 to 100 proxies and 5000 assignments each.

4.4 Bystander Results

Load balancing and the proportion of attackers directly affect the number of bystanders. Bystanders are those honest clients that still receive service in the presence of malicious clients. For the experiment in Figure 4.6, we run all of the algorithms so they each have 5000 assignments and the probability that a client is malicious is 50%. We sort the proxies by their loads in increasing order and plot the numbers of malicious clients assigned to each proxy, from lowest loaded proxy to highest loaded.

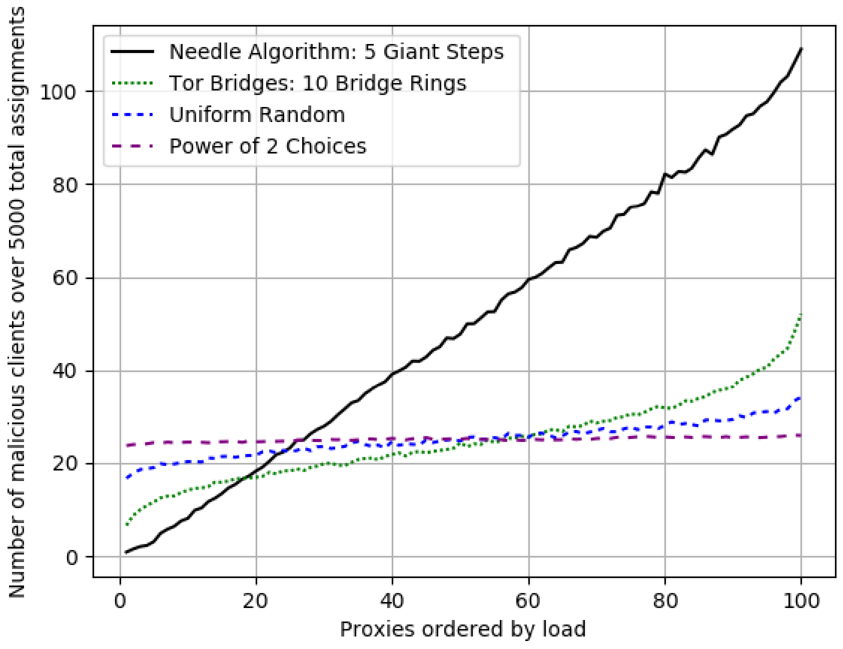

Figure 4.6: Comparison of Malicious Clients over 5000 assignments with 50% attackers in the system.

We see from the results of this experiment that the number of malicious clients increases with load in all of the algorithms. The power of 2 choices algorithm has the flattest distribution of malicious clients, meaning that proxies are enumerated the quickest thus creating the highest number of enumerated proxies over all of the algorithms, as we discussed in the enumeration analysis in section 3.2.

When we consider the number of bystanders, we make the assumption that the censor chooses to block the top half of the proxies. These are the proxies that are determined as the most *popular* proxies. Figure 4.7 shows the total number of bystanders that receive service in each algorithm, assuming that the most heavily loaded $n/2$ proxies are blocked; that is, the rightmost $n/2$ proxies in Figure 4.6. The steep slope in the needle's load balancing for the least heavily loaded, leftmost

$n/2$ unblocked proxies, allows for a small number of proxies to be preserved, while stacking higher loads onto the remaining proxies. It also serves to distinguish the rightmost proxies as more popular than the others, in a sense serving up these proxies to be blocked as expected collateral damage.

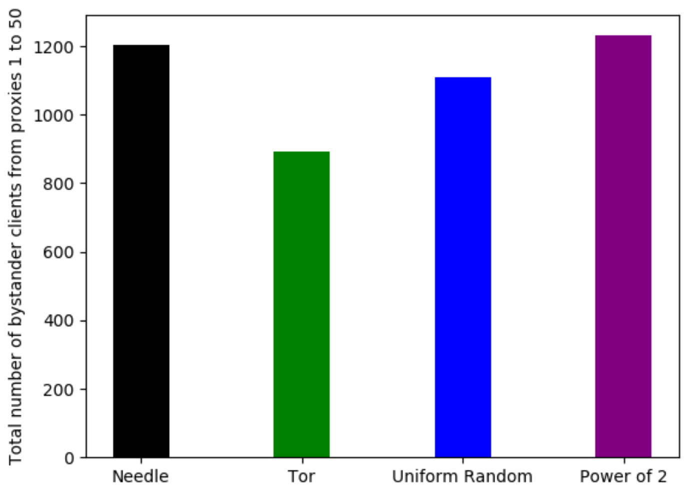

Figure 4.7: Total number of bystander clients in the least popular proxies.

We've shown that by unbalancing the load on the proxies in the needle algorithm, we are still able to service as many or more bystander clients than in the uniform random, power of 2 choices, and Tor's bridgedb mechanism.

4.5 Comparison

We consider enumeration, load balancing, and bystander analyses of the four algorithms in order to contrast their respective trade-offs and suitability under differing system goals. Table 4.3 outlines the three metrics that we consider in this thesis. We indicate with checkmarks ✓ where the algorithm is well suited, double checkmarks indicate that it is very well suited. A single X mark ✗ shows that the algorithm is unsuitable, a double ✗ is used when we consider the algorithm to be completely unusable for a purpose.

	Enumeration	Load Balancing	Bystanders
power of 2 choices	✗✗	✓✓	✓✓
uniform random	✗	✓	✗
Tor's `bridgedb`	✗	✓	✗
needle	✓✓	✗✗	✓✓

Table 4.3: Comparison chart of the 4 algorithms

The power of 2 choices algorithm provided the best load balancing, and so would be appropriate for systems where this is a concern. However, it is enumerated very quickly.

Uniform random distribution and Tor's `bridgedb` distribution perform similarly, and so we consider their respective trade-offs in the same vein. They provide a slower enumeration time than the power of 2 choices algorithm due to their uneven load balancing. The number of bystanders is also slightly lower than the other algorithms.

The needle algorithm gives the slowest enumeration of all the algorithms; this attribute is beneficial to allocation of proxies in a distribution system. It is not suitable for systems that require even load balancing. (It is unbalanced to a large degree but not so much that it is unusable.) It services as many bystanders as the power of 2 choices algorithm, but without the drawback of fast enumeration. In the context of proxy distribution under our censorship threat model, we consider this a benefit.

Chapter 5

Related Work

Censorship resistance systems CRS are composed of two parts; 1) communication establishment, or handshake, needed to join the CRS, and 2) the conversation stage where the actual information is exchanged between the client and the censored site. In *SOK: Making sense of censorship resistance systems* [14], categorizations of different types of CRS are outlined based on the system's approach to these two functions.

There are two main approaches to communication establishment in censorship resistance systems; resource scheduling and allocation approach, and the trust-based approach. Resource scheduling and allocation fight a losing battle to distribute proxies under a censorship threat model where proxies are continuously blocked. Proxies need to be birthed or placed into reserve in order to keep up with the censor's blocking rate. The advantage of these schemes is that they are lightweight and easy to implement.

The goal of trust-based proxy distribution is to mitigate blocking by building trust between honest users of the system and the proxy distributor. They achieve this by distinguishing honest clients from malicious clients. We outline systems and techniques that fall under a similar censorship threat model as our own, where the censor is omnipotent and blocks proxies based on information gained by posing as an honest user.

Note that other types of anonymous systems, such as Vuvuzela [24], Dissent [8], and Freenet [7], are concerned with maintaining levels of anonymity in order to prevent an adversary from learning about messages from a sender to a receiver. The highest level of anonymity in these systems is a *third axis* where one cannot tell that a user is communicating with any other user [23]. The adversary in the proxy distribution problem under a censorship threat model is vastly stronger than in general anonymity systems. The goal of CRS is to obfuscate evidence that a user is in the system at all, as well as to hide the activity of the user within said system.

5.1 Resource Scheduling and Allocation

Tor's Bridge Distribution. Tor bridges are private relays or proxies used for censorship circumvention. Tor bridge IP addresses and fingerprints are distributed out of band using registered email or through a captcha site on the Tor blog. Tor's BridgeDB authority distributes up to 3 new bridge IPs and corresponding fingerprints to clients based on a hashring uniform distribution. Bridge requests are rate limited by a centralized bridge distributor. Despite these efforts, censors in China have discovered and blocked most of the bridges given through the public distribution channels. Bridge enumeration attacks are possible using bulk emails via HTTPS [16]. Tor uses a fingerprint as a shared secret scheme to thwart active probing, however this can't prevent bridge discovery using a delayed insider attack [10].

TorBricks. The TorBricks [28] design distributes proxies in groups to guarantee a maximum number of rounds until all honest users can connect to a proxy server after some number of retries. It relies on exponential growth in the number of proxies in order to provide these guarantees of a logarithmic number of rounds. A caveat in this system is, if there are no unblocked proxies in the current group, then the algorithm requires that a unique bridge be allocated per each user.

While this approach allows for adaptive adjustment to a proportion of attackers in the system, it requires a great deal of resources. For example, if a CRS had enough proxies to give one out per each user, we would not have the problem of proxy distribution since we could hand out private proxies. If we have private proxies, then no attacker could discover an honest user's proxy because users would not have to share.

Fighting Censorship with Algorithms. Mahdian [17] studies proxy distribution as an algorithmic problem and gives bounds on the number of proxies required to provide service to clients, some of whom are adversaries. He includes a theoretical analysis of bounds for the number of proxies needed to survive an insider attack. His theorems use k-union-free families of sets, probabilistic methods, and extremal set theory to give lower and upper bounds.

Mahdian's scheme creates two sets of users, trusted and suspicious, and distributes keys based on the user's membership in one of these two sets. Users are divided into these sets based on their association with compromised keys; they are moved from the trusted group to the suspicious group. Fresh keys are only handed out to trusted users. This adaptive model bounds the number of keys that an adversary can compromise thus providing guarantees for the user with respect to the expected total number of keys required to give every legitimate user access.

In Mahdian's model, a key represents a servlet or proxy server. His scheme assumes a known number of malicious users and there are no bounds on the number of clients a proxy can serve. While maintenance of keys is usually relatively simple in practice, the logistics of proxy server maintenance is more involved. It is an impractical assumption that keys (representing proxies) can be distributed to an unlimited number of users without significant overhead.

Mahdian's algorithm distributes an increasing number of keys to users in order to reduce the risk posed by a growing number of adversaries. This does not address the case where adversaries are controlled by the same entity, such as a censor. For example, it does not take into consideration the enumeration attack. This attack is further exacerbated by the reuse of keys, as opposed to the removal of suspect keys.

5.2 Trust-based Allocation

Proximax. The main goal of the Proximax [18] reputation-based system is to maximize the yield of a proxy resource, where yield is the number of user-hours per day before a proxy is blocked, calculated as the product of usage and lifetime. Each proxy resource is advertised on multiple channels. This novel use of a channel relies on a fast flux technique that piggybacks on Domain Name System (DNS) infrastructure. Proximax registers multiple proxies to the same domain name and load balances them based on their current utilization and resource risk parameter.

The usage and risk of a proxy resource is the sum of the risk of each of the channels where it is distributed. Resource risk is calculated as a maximum likelihood estimate of blocking - it is only an approximation because resources are advertised on multiple channels, and the risk per channel cannot be sampled directly. In other words, when a proxy is blocked, there is no way to detect the specific channel that caused the block and so it is difficult to tease out channel from proxy risk.

All proxies will eventually be discovered in proxy distribution, so Proximax adds a trust scheme to delay censor discovery. Registered users build up their reputation score and invite new users, handled by a registration system. The registration system allocates proxies that have higher risk to lower reputation scores. They widely disseminate the location of low risk proxies in order to maximize their yield. This leads to the potential of rapid enumeration attacks, where the best proxies are enumerated in quick succession.

Proximax's trust scheme is likely to be thwarted by colluding insider attackers where registered users build up reputation to invite other users, as it does not work well in a delayed blocking attack. Furthermore, the risk approximation may not be particularly useful because if there is only a single attacker assigned to a proxy, and with the reasonable assumption that all attackers are controlled by the same censor entity, then this proxy has the same likelihood of being blocked as a proxy to which several insider attackers are assigned.

rBridge. rBridge [25] is a trust-based reputation system for a Tor bridge distributor that addresses insider attacks, minimizes user wait time for an available proxy, and preserves privacy of client assignment information. The rBridge distributor computes user reputation based on the uptime of bridges to which a user is assigned. A payment system allows users to buy unblocked bridges to prevent repeated blocks. They show rBridge's user-hours served is at least one order of magnitude more than Proximax and that thirsty hours of users waiting for a proxy is minimized. This is mainly achieved by making sure that the overall rate of new bridges outpaces the rate of proxy blocks, and by reserving half of their bridge resources for future invitations.

A significant contribution of rBridge's design is their privacy preserving scheme using anonymous credentials to build trust, invite users, and obtain signed credentials. Restricting proxy assignments can lead to user fingerprinting as there are unique combinations of proxies tied to a single user. Previous work in pseudonymous credentials and oblivious transfer methods don't work well in proxy systems because an attacker can still infer client assignments based on behaviour after a proxy is blocked. rBridge hides bridge assignment from even the distributor by enabling the proxy assignments to be written and updated by users. They take extra measures to maintain integrity using anonymous credentials, one-time tokens and secrets that cannot be forged owing to zero knowledge proofs and blind signatures.

In addressing the delayed blocking attack, they note that invitation tickets are randomly distributed over all users, so there is a chance that the corrupt user may not receive a ticket. Since tickets cannot be transferred, it is no more likely that an attacker receives a ticket than an honest user. However, they do not provide analysis given that even just one corrupt user is more than enough to block a proxy, therefore an assignment of a proxy to a single attacker is more significant than assignment to an honest user.

5.3 Routing

Routing users through decoy paths as a mechanism to discover proxies is a vastly different approach to the proxy distribution problem and handshake used in censorship resistance systems. Essentially, this approach solves the issue of censors posing as honest users and blocking proxies because censors do not want to block the decoy paths. This is because there would be too much collateral damage as the decoy paths are heavily used and it is difficult to tell if users are accessing blocked sites from the decoys. The barrier to implement these solutions, however, is that they require a large commitment from either Content Delivery Networks (CDN) or Internet Service Providers (ISP) to build the decoy paths and to maintain the network that provides the anonymous handshake.

Telex. This CRS uses destination obfuscation via decoy routing at Telex stations that are *end-to-middle* proxies located in their own network infrastructure [26]. Sessions between users and Telex have special tags to direct them through to censored sites. These proxies are built into the network with involvement from local ISP that must deploy Telex stations on paths between networks under censor control and blocked destinations.

Domain Fronting. An elegant way to circumvent censors is through domain fronting, where a user is routed through a legitimate intermediary, such as a CDN [9]. These intermediaries are used to rendezvous with Tor and cannot be detected by a censor because the CDN's network is beyond the censor boundary. Domain fronting is the most reliable way to perform the rendezvous handshake and recently it is the only protocol that works in China [4]. However, we see that major CDN like Google and Amazon no longer support domain fronting. Microsoft's Azure cloud is a temporary fix as there is no formal agreement between Microsoft and Tor to support domain fronting in the future.

5.4 Related Approaches

The following works aren't directly related to the problem of censorship circumvention. However, their analysis of resource allocation is closely tied to the coupon collector analysis in this thesis. Their lightweight distribution under a different threat model also relates to and inspired our approach.

Coupon Collector and Power of d Choices. The Power of d Choices was analyzed for the generalized form of the coupon collector problem in [27]. This includes the case where a collector wants to collect m out of n total coupons. The collector selects d coupons out of the total collection and chooses the least heavily loaded (or least collected) coupon in each draw. This benefits the collector because duplicate coupons are discarded. They show that the expected number of draws to collect m out of a total of n coupons is $(n \log n)/d + (n/d)(m1) \log\log n + O(mn)$. Although this is opposite from our reverse power of 2 choice algorithm, and far more complex in its analyses, it is a useful counterpoint to our goal of delaying enumeration, as the goal here is to enable the coupon collector to collect coupons as quickly as possible.

Proxisch. Proxisch [13] is an application of a scheduling algorithm for proxy distribution in web crawling applications. Although Proxisch is intended for distributed web crawling, the proxy server selection mechanism has a similar goal shared by the proxy distribution problem; to reduce the risk of assigning high risk proxies to clients. This work estimates the reliability of proxies based on a reliability calculation and uses queuing theory to organize proxies by their respective reliability factors.

To model the life of a proxy, they use an exponential distribution based on the life of an ideal lamp. They show in their simulations that the actual life span of some proxy servers is close to the exponential distribution. This leads to the calculation of an optimal update period for cycling proxies among processes. They compare their result to a polling scheduling solution that illustrates higher successful service rates within shorter time periods for their solution. Their resource scheduling algorithm approach is similar to a randomized proxy selection based on attributes of proxies, such as time, reputation, or credits because it orders proxies based on criterion. This is a move away from more complex, monolithic designs favouring proxies that earn standing over time within the system.

It's not possible to directly translate this solution into the problem addressed in this thesis, since modeling the lifetime of a proxy server does not directly translate to the motivations of a powerful censor that can cut the lifetime of a proxy at any time. However, this work does provide useful hints for how one may approach a lightweight proxy distribution design.

Chapter 6

Discussion

6.1 Future Work

Blocking behaviour. The simulation can easily extend to model different forms of blocking behaviour by a censor. The parameters for a blocking rate process are coded in the simulation although these were not utilized in the simulation nor analyzed for the evaluation.

Proxies joining and leaving. A more realistic version of the simulator is to run the experiments with some proxies joining and leaving. This would provide more data on how the needle algorithm preserves new proxies, rather than only dealing with proxies that are created at the same time.

Service times. The simulation was not fully utilized to analyze classic problems like Quality of Service. It would be particularly interesting to examine how non-needle proxies are able to handle the larger loads in a real system.

Tor integration. The needle algorithm could be run inside of the Tor `BridgeDB` codebase to validate the simplicity of the algorithm's approach. It would require a different method of client assignment because Tor uses hashes of IP addresses and does not store load or assignment information in the same way that the needle algorithm requires for its operation.

Chapter 7

Conclusion

Our work has the same goal as many proxy distribution systems; to provide service to clients in a hostile environment with practically an omnipotent censor entity. Our trust-less, elegant approach to proxy distribution relies on essentially hiding *needle* proxies in distribution rounds.

Building trust requires storage of user behaviour over time. Within anonymous systems, the assumption that a system itself can be trusted to store this information is an extremely complex topic, both technically and ideologically. It may be argued that any proxy system provides some measure of anonymity guarantees because many users are aggregated onto single proxies. However, there exists a fundamental distrust of any proxy distribution system that persists user information.

Many trust-based systems are proposed yet few are adopted in practice. Simpler systems such as lightweight proxy distribution and decoy routing are adopted widely, and it may not only be a result of their implementation simplicity. Perhaps, the foundation of trust-based systems goes against the ethos of anonymity. As our privacy concerns grow, we look for more varieties of systems with demonstrable privacy guarantees that align with our privacy needs. The wider the adoption of such systems, the more privacy guarantees we can give. The more honest users in the system, the less of an impact that attackers can have. It requires the effort of everyone to hide vulnerable parties in the crowd. If we volunteer our resources and participate in the privacy community, it is possible.

Bibliography

[1] *Research problems: Ten ways to discover Tor bridges*, 2019 (accessed April 9, 2019). URL https://blog.torproject.org/research-problems-ten-ways-discover-tor-bridges. → page 6

[2] *BridgeDB*, 2019 (accessed April 9, 2019). URL https://bridges.torproject.org. → page 2

[3] *OONI Project*, 2019 (accessed April 9, 2019). URL https://blog.torproject.org/tor-heart-ooni-project. → page 2

[4] *Domain Fronting Is Critical to the Open Web*, 2019 (accessed April 9, 2019). URL https://blog.torproject.org/domain-fronting-critical-open-web. → page 45

[5] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. *SIAM journal on computing*, 29(1):180–200, 1999. → page 9

[6] P. Berenbrink, A. Czumaj, A. Steger, and B. Vocking. Balanced allocations: the heavily loaded case. In *Annual ACM Symposium on Theory of Computing: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, volume 21, pages 745–754, 2000. → page 10

[7] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing privacy enhancing technologies*, pages 46–66. Springer, 2001. → page 40

[8] H. Corrigan-Gibbs and B. Ford. Dissent: accountable anonymous group messaging. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 340–350. ACM, 2010. → page 40

[9] D. Fifield. *Threat modeling and circumvention of Internet censorship*. PhD thesis, University of California, Berkeley, Dec. 2017. URL https://www.bamsoftware.com/papers/thesis/. → pages 2, 45

[10] D. Fifield and L. Tsai. Censors' delay in blocking circumvention proxies. *arXiv preprint arXiv:1605.08808*, 2016. → page 41

[11] P. Flajolet, D. Gardy, and L. Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39(3):207–229, 1992. → pages 4, 7, 8

[12] G. H. Gonnet. Expected length of the longest probe sequence in hash code searching. *Journal of the ACM (JACM)*, 28(2):289–304, 1981. → page 9

[13] X. Jiang, J. Shi, Q. Tan, W. Zhang, X. Wang, and M. Chen. Proxisch: An optimization approach of large-scale unstable proxy servers scheduling. *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 10(6):1149–1154, 2016. → page 46

[14] S. Khattak, T. Elahi, L. Simon, C. M. Swanson, S. J. Murdoch, and I. Goldberg. Sok: Making sense of censorship resistance systems. *Proceedings on Privacy Enhancing Technologies*, 2016(4):37–61, 2016. → page 40

[15] Y. A. Lateef. *Repository of scales and melodic patterns*. Fana Music, 1981. → page 12

[16] Z. Ling, J. Luo, W. Yu, M. Yang, and X. Fu. Tor bridge discovery: extensive analysis and large-scale empirical evaluation. *IEEE Transactions on Parallel and Distributed Systems*, 26(7):1887–1899, 2015. → pages 11, 41

[17] M. Mahdian. Fighting censorship with algorithms. In *International Conference on Fun with Algorithms*, pages 296–306. Springer, 2010. → page 42

[18] D. McCoy, J. A. Morales, and K. Levchenko. Proximax: A measurement based system for proxies dissemination. *Financial Cryptography and Data Security*, 5(9):10, 2011. → page 42

[19] M. Mitzenmacher and E. Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005. → pages 8, 9

[20] M. D. Mitzenmacher. *The power of two random choices in randomized load balancing*. PhD thesis, PhD thesis, Graduate Division of the University of California at Berkley, 1996. → pages 9, 31

[21] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge university press, 1995. → page 6

[22] A. N. Myers and H. S. Wilf. Some new aspects of the coupon collector's problem. *SIAM review*, 48(3):549–565, 2006. → page 8

[23] M. K. Reiter and A. D. Rubin. Anonymous web transactions with crowds. *Communications of the ACM*, 42(2):32–48, 1999. → page 41

[24] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 137–152, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3834-9. doi:10.1145/2815400.2815417. URL http://doi.acm.org/10.1145/2815400.2815417. → page 40

[25] Q. Wang, Z. Lin, N. Borisov, and N. Hopper. rbridge: User reputation based tor bridge distribution with privacy preservation. In *NDSS*, 2013. → pages 2, 5, 43

[26] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman. Telex: Anticensorship in the network infrastructure. In *USENIX Security Symposium*, 2011. → page 45

[27] W. Xu and A. K. Tang. A generalized coupon collector problem. *Journal of Applied Probability*, 48(4):1081–1094, 2011. → pages 33, 45

[28] M. Zamani, J. Saia, and J. Crandall. Torbricks: Blocking-resistant tor bridge distribution. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 426–440. Springer, 2017. → page 41