TorBricks: Blocking-Resistant Tor Bridge Distribution

Mahdi Zamani
Yale University
New Haven, CT
mahdi.zamani@yale.edu

Jared Saia
University of New Mexico
Albuquerque, NM
saia@cs.unm.edu

Jedidiah Crandall
University of New Mexico
Albuquerque, NM
crandall@cs.unm.edu

ABSTRACT

Tor is currently the most popular network for anonymous Internet access. It critically relies on volunteer nodes called *bridges* for relaying Internet traffic when a user's ISP blocks connections to Tor. Unfortunately, current methods for distributing bridges are vulnerable to malicious users who obtain and block bridge addresses. In this paper, we propose TorBricks, a protocol for distributing Tor bridges to n users, even when an unknown number $t < n$ of these users are controlled by a malicious adversary. TorBricks distributes $O(t \log n)$ bridges and guarantees that all honest users can connect to Tor with high probability after $O(\log t)$ rounds of communication with the distributor.

We also extend our algorithm to perform privacy-preserving bridge distribution when running among multiple untrusted distributors. This not only prevents the distributors from learning bridge addresses and bridge assignment information but also provides resistance against malicious attacks from a $\lfloor m/3 \rfloor$ fraction of the distributors, where m is the number of distributors.

1. INTRODUCTION

Mass surveillance and censorship increasingly threaten democracy and freedom of speech. A growing number of governments around the world control the Internet to protect their domestic political, social, financial, and security interests [?, ?]. Countering this trend is the rise of anonymous communication systems, which strive to preserve the privacy of individuals in cyberspace. Tor [?] is the most popular of such systems with more than 2.5 million users on average per day [?]. Tor relays Internet traffic via more than 6,500 volunteer nodes called *relays* spread across the world [?]. By routing data through random paths in the network, Tor can protect the private information of its users such as identity and geographical location.

Since the list of all relays is available publicly, state-sponsored organizations can enforce Internet service providers (ISPs) to block access to all of them making Tor unavailable in territories ruled by the state. When access to the Tor network is blocked, Tor users have the option to use *bridges*, which are volunteer relays not listed in Tor's public directory [?]. Bridges serve only as entry points into the rest of the Tor network, and their addresses are carefully distributed to the users, with the hope that they cannot all be learned by censors. As of March 2016, about 3,000 bridge nodes were running daily in the Tor network [?].

Currently, bridges are distributed to users based on different strategies such as CAPTCHA-enabled email-based distribution and IP-based distribution [?]. Unfortunately, censors are using sophisticated attacks to obtain and block bridges, rendering Tor unavailable for many users [?, ?, ?]. Also, state of the art techniques for bridge distribution either (1) cannot provably guarantee that all honest users can access Tor [?, ?, ?]; (2) can only work when the number of corrupt users is known in advance [?]; (3) require fully trusted distributors [?, ?, ?]; and/or (4) cannot resist malicious attacks from the distributors [?, ?, ?, ?].

We believe that it is challenging in practice to identify or even estimate the number of corrupt users, due to the sophisticated nature of Internet censorship in many countries such as China [?, ?]. Moreover, we believe it is desirable for a system to be robust to attacks on the distributors. For example, powerful adversaries can corrupt these systems, not only to break users' anonymity by obtaining bridge assignment information but also to prevent the bridge distribution protocol from achieving its goals.

In this paper, we propose TorBricks, a bridge distribution algorithm that provably ensures Tor is available to all honest users with high probability, without requiring any *a priori* knowledge about the number of corrupt users. TorBricks guarantees that the number of rounds until all honest users can connect to Tor is bounded by $O(\log t)$, where $t < n$ is the number of corrupt users. This is achieved by distributing at most $O(t \log n)$ bridge addresses, where n is the total number of users.

We also describe a privacy-preserving bridge distribution mechanism for the scenarios where a certain fraction of the distributors may be controlled maliciously by the adversary. We stress that TorBricks can run independently from Tor so that the Tor network can focus on its primary purpose of providing anonymity.

The rest of this paper is organized as follows. In Section 1.1, we describe our network and threat model. In Section 1.2, we state our main result as a theorem. We review related work in Section 2. In Section 3, we describe our algorithms for reliable bridge distribution; we start from a simple algorithm and improve it as we continue. We describe our implementation of TorBricks and our simulation results in Section 4. Finally, we summarize and state our open problems in Section 5.

1.1 Network and Threat Model

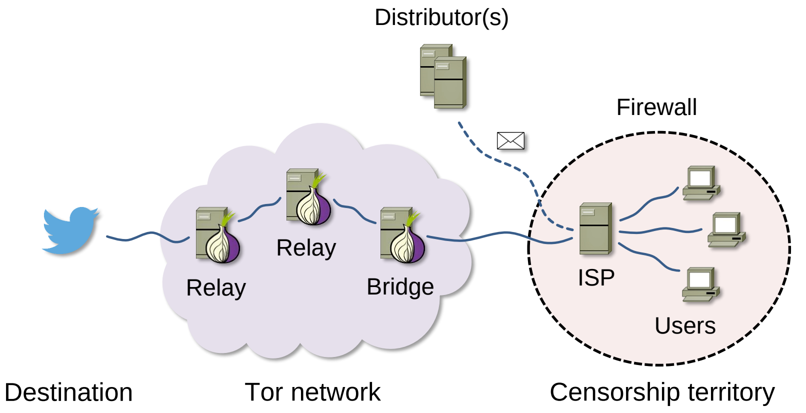

Figure 1: Our network model

In this section, we first define a basic model, where a single distributor performs the bridge distribution task, and then define a multiple distributors model, where a group of distributors collectively run our algorithm. Figure 1 depicts our high-level network model.

Basic Model. We assume there are n *users* (or *clients*) who need to obtain bridge addresses to access Tor. Initially, we assume a single trusted server called the *bridge distributor* (or simply the *distributor*), which has access to a reliable supply of bridge addresses.

We assume an adversary (or *censor*) who can view the internal state and control the actions of t of the clients; we call these adversarially-controlled clients *corrupt users*. The adversary is *adaptive* meaning that it can corrupt users at any point of the algorithm, up to the point of taking over t users.

The corrupt users have the ability to *block* bridges whose IP addresses they receive. A bridge that is blocked cannot be used by any user. The adversary does not have to block a bridge as soon as it finds its address; he is allowed to strategically (perhaps by colluding with other corrupt users) decide when to block a bridge. We refer to the other $n - t$ users as *honest users*. Each honest user wants to obtain a list of bridge addresses, at least one of which is not blocked and hence can be used to connect to Tor. We further assume that the adversary has no knowledge of the private random bits used by our algorithm.

We make the standard assumption that there exists a rate-limited channel such as email that the users can use to send their requests for bridges to the distributor, but which is not suitable for interactive Internet communication such as web surfing.[1] The distributor runs our bridge distribution algorithm locally and sends bridge assignments back to the users via the same channel.

Multiple Distributors Model. We also consider a *multiple distributors model*, where a group of $m \ll n$ distributors collectively distribute bridge addresses among the users such that none of the distributors can learn any information about the user-bridge assignments. We assume that the distributors are connected to each other pairwise via a synchronous network with reliable and authenticated channels.

In this model, the adversary not only can corrupt an unknown number of the users, t but can also maliciously control and read the internal state of up to $\lfloor m/3 \rfloor$ of the distributors. The corrupt distributors can deviate from our protocols in any arbitrary manner, e.g., by sending invalid messages or remaining silent. We assume that the bridges also use the rate-limited channel for communicating with the distributors with the purpose of registering themselves in the system. Figure 3 shows our multiple distributors model.

Testing Bridge Reachability. We assume the distributors obtain the availability information of bridges (i.e., which bridges are blocked and which ones are not) from the Tor network. This assumption relies on a technique for testing reachability of bridges from outside the censored territory. Recent work [**?**, **?**] describe active scanning mechanisms that can be used to test the reachability of bridges in real time with low latency, and we expect that one of these mechanisms to be deployed on the Tor network in the near future. The details of these methods and their current challenges are out of the scope of our paper.

1.2 Our Result

Below is our main theorem, which we prove in Section 3.

THEOREM 1. *There exists a bridge distribution protocol that can run among m distributors and guarantee the following properties with probability $1 - 1/n^\kappa$, for some constant $\kappa \geq 1$, in the presence of a malicious adversary corrupting at most $\lfloor m/3 \rfloor$ of the distributors:*

1. *All honest users can connect to Tor after $\lceil \log \lceil (t+1)/32 \rceil \rceil + 1$ rounds of communication with the distributors;*
2. *The total number of bridges required is $O(t \log n)$;*
3. *Each user receives m messages in each round;*
4. *Each distributor sends/receives $O(m^2 + n)$ messages;*
5. *Each message has length $O(\log n)$ bits.*

[1] Completely blocking a service such as email would usually impose significant economic and political consequences for censors.

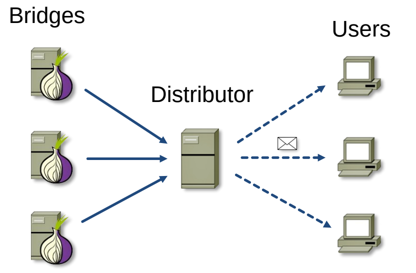

Figure 2: Single distributor model

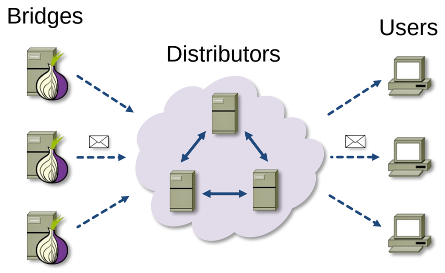

Figure 3: Multiple distributors model

We also implement a prototype of TorBricks and conduct simulations to measure the running time and bridge cost of the protocol. We discuss our experimental results in Section 4.

1.3 Technical Challenges

The key technical novelties in the design of TorBricks are as follows:

1. Resource-Competitive Costs. Our protocol adaptively increases the number of bridges distributed among the users with respect to the number of bridges recently blocked by the adversary. This reduces the number of bridges used by the algorithm at the expense of a small (logarithmic) latency cost, which is also a function of the adversary's cost. As a result, the overhead of TorBricks will always be proportional to the amount of corruption by the adversary. The details are described in Section 3.1.

2. Handling Client Churn. In practice, users join and leave the system frequently. Adding new users to the system while offering provable robustness against an unknown number of corrupt users is challenging. This is because either (1) the adversary can cause a denial of service to the new users if the algorithm's "next move" is based on the adversary's behavior, or (2) the protocol cannot guarantee every user receives a usable bridge. We propose a simple technique to handle this with small (constant) latency overhead. The details are described in Section 3.1.2.

3. Oblivious Bridge Distribution. Our technique for computing user-bridge assignments does not depend on actual bridge addresses. Thus, the distributor can assign "bridge pseudonyms" to the users. This prevents an honest-but-curious distributor from snooping on the user-bridge assignments. We also show how to distribute these pseudonyms among a group of geographically-dispersed servers who can collectively give the users the information needed to reconstruct their bridge addresses. This protects the anonymity of the users against colluding distributors. The details are described in Sections 3.2.1 and 3.2.2.

4. Distributed Random Generation. We show how a *distributed random generation (DRG)* protocol can be used to solve the bridge distribution problem efficiently among multiple untrusted distributors. A DRG protocol allows a group of nodes to generate a uniform random number collectively in a way that none of the nodes can learn it before others, or bias it from the uniform distribution. We employ a well known DRG protocol in TorBricks to provide the first bridge distribution mechanism that can resist malicious (active) attacks from a subset of the distributors. The details are described in Section 3.2.2.

2. RELATED WORK

Proxy Distribution. The bridge distribution problem has been studied as the *proxy distribution*, where a set of proxy servers outside a censorship territory are distributed among a set of users inside the territory. These proxies are used to relay Internet traffic to blocked websites.

Feamster et al. [**?**] propose a proxy distribution algorithm that requires every user to solve a cryptographic puzzle to discover a proxy. In this way, the algorithm prevents corrupt users from learning a large number of proxies. Unfortunately, empirical results of [**?**] show that a computationally powerful censor can easily block a large fraction of the proxies.

The Kaleidoscope system of Sovran et al. [**?**] disseminates proxy addresses over a social network whose links correspond to existing real world social relationships among users. Unfortunately, this algorithm assumes the existence of a few internal trusted users who can relay other users' traffic. Also, Kaleidoscope cannot guarantee its users' access to Tor.

McCoy et al. [**?**] propose Proximax; a proxy distribution system that uses social networks such as Facebook as trust networks that can provide a degree of protection against discovery by censors. Proximax estimates each user's effectiveness, and chooses the most effective users for advertising proxies, with the goals of maximizing the usage of these proxies while minimizing the risk of having them blocked.

Mahdian [**?**] studies the proxy distribution problem when the number of corrupt users, t, is known in advance. He proposes algorithms for both large and small values of t and provides a lower bound for dynamic proxy distribution that is useful only when $t \ll n$. Unfortunately, it is usually hard in practice to reliably estimate the value of t. Mahdian's algorithm for large known t requires at most

$t\,(1 + \lceil \log{(n/t)} \rceil)$ bridges, and his algorithm for small known t uses $O(t^2 \log n / \log \log n)$ bridges.

Wang et al. [?] propose a reputation-based bridge distribution mechanism called rBridge that computes every user's reputation based on the uptime of its assigned bridges and allows the user to replace a blocked bridge by paying some reputation credits. Interestingly, rBridge is the first model to provide user privacy against an honest-but-curious distributor. This is achieved by performing oblivious transfer between the distributor and the users along with commitments and zero-knowledge proofs for achieving unlinkability of transactions.

Handling DPI and Active Probing. The Tor Project has developed a variety of tools known as *pluggable transports* [?] to obfuscate the traffic transmitted between clients and bridges. This makes it hard for the censor to perform *deep packet inspection (DPI)* attacks, since distinguishing actual Tor traffic from legitimate-looking obfuscated traffic is hard.

The censor can also block bridges using *active probing*: he can passively monitor the network for suspicious traffic, and then actively probe dubious servers to block those that are determined to run the Tor protocol [?]. We believe that active probing will be defeated in the future using a combination of ideas from CAPTCHAs, port knocking [?], and format transforming encryption [?]. Depending on the sophistication of the censor, TorBricks may be used in parallel with tools that can handle DPI and active probing to provide further protection against blocking.

Resource-Competitive Analysis. Our analytical approach to bridge distribution can be seen as an application of the *resource-competitive analysis* introduced by Gilbert et al. [?, ?]. This approach evaluates the performance of any distributed algorithm under attack by an adversary in the following way: if the adversary has a budget of t, then the worst-case resource cost of the algorithm is measured by some function of t. The adversary's budget is frequently expressed by the number of corrupt nodes controlled by the adversary. This model allows the system to adaptively increase/decrease its resource cost with the *current* amount of corruption by the adversary. Inspired by this model, we design resource-competitive algorithms for bridge distribution that scale reasonably with the adversary's budget.

3. OUR ALGORITHMS

We first construct a bridge distribution algorithm that is run locally by a single distributor. In Section 3.2, we extend this algorithm to the multiple distributor model. We prove the desired properties of these algorithms in Section 3.1.1 and Section 3.2 respectively. Before proceeding to our algorithms, we define standard terms and notation used in the rest of the paper.

Notation. We say an event occurs *with high probability*, if it occurs with probability at least $1 - 1/n^{\kappa}$, for some constant $\kappa \geq 1$. We denote the set of integers $\{1, ..., n\}$ by $[n]$, the natural logarithm of any real number x by $\ln x$, and the logarithm to the base 2 of x by $\log x$. We denote a set of n users participating in our algorithms by $\{u_1, ..., u_n\}$. We define the *latency* of our algorithm as the maximum number of rounds of communication that any user has to perform with the distributor(s) until he obtains at least one unblocked bridge.

3.1 Basic Algorithm

Our basic algorithm (shown in Algorithm 1) is run locally by one distributor.[2] The algorithm proceeds in *rounds*, where each round corresponds to an increment of the variable i in the while loop. In each round, the algorithm assigns a set of bridges randomly to a fixed group of users and proceeds to the next round only if the number of blocked bridges exceeds a threshold that is increased in each round.

The number of bridges distributed in every round is determined based on the threshold in that round as depicted in Figure 4. If the number of bridges to be distributed in the current round becomes larger than the number of users, n, then the algorithm simply assigns a unique unblocked bridge to every user. This happens only if the adversary blocks a significant number of bridges and we believe it does not happen in most practical cases. If it happens, though, it becomes more reasonable for the algorithm to give each user a unique user.

The exponential growth of the number of bridges distributed in each round allows us to achieve a logarithmic number of rounds (on t) until all users can connect to Tor with high probability. In Lemma 3, we calculate the exact number of rounds required to achieve this goal.

We later show that if one instance of Steps 1–23 of Algorithm 1 is run, then it guarantees that all users can connect to Tor with some constant probability. Therefore, if we repeat these steps $3 \log n$ times, then we can guarantee that all users can connect to Tor with high probability. If the $3 \log n$ instances run completely independently, then the adversary can take advantage of this to increase the latency of the algorithm by a factor of $3 \log n$ using a *serialization attack*. In this attack, the adversary can strategically coordinate with its corrupt users to block the assigned bridges in such a way that the instances proceed to the next round one at a time. We prevent this attack by maintaining a single round counter, i, for all instances: whenever the number of blocked bridges in *any* of the instances exceeds the threshold for the current round (i.e., $0.6 \times 2^{i+4}$), all instances proceed to the next round.

In every round, TorBricks only distributes unblocked bridges. To minimize the total number of bridges required, we use unblocked bridges from previous rounds in the current distribution round. This can be done by removing blocked bridges from the pile of previously recruited bridges and adding a sufficient number of new bridges to accommodate the new load.

In each round, TorBricks sends to every user a single message containing $3 \log n$ bridge addresses assigned to this user in all instances of Algorithm 1 that run in parallel. This message is sent to the user via the rate-limited channel (e.g., email).

3.1.1 Analysis of Algorithm 1

[2] By "run locally", we mean the distributor computes user-bridge assignments independent of any other distributor and without exchanging any information with them.

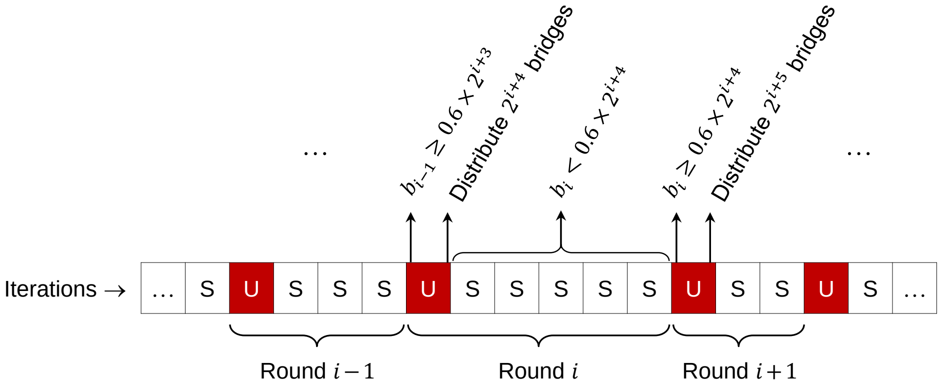
…
$b_{i-1} \geq 0.6 \times 2^{i+3}$
Distribute 2^{i+4} bridges
$b_i < 0.6 \times 2^{i+4}$
$b_i \geq 0.6 \times 2^{i+4}$
Distribute 2^{i+5} bridges
…
Iterations →
… S U S S S U S S S S S U S S U S …
Round $i-1$
Round i
Round $i+1$

Figure 4: Number of bridges distributed in round i of Algorithm 1. S and U indicate successful and unsuccessful rounds.

We now prove that Algorithm 1 achieves the properties described in Theorem 1 in the single distributor model. We assume a user can connect to Tor in an iteration of the while loop if and only if at least one unblocked bridge is assigned to it. Although the adversary has a total budget of t corrupt users, only some of the corrupt users might be actively blocking bridges in any given round. Before stating our first lemma, we define the following variables:

- b_i: number of bridges blocked in round i.
- d_i: number of bridges distributed in round i.
- t_i: number of corrupt users each of whom has blocked at least one bridge in round i.

Lemma 1 (Robustness). *In round i of Algorithm 1, if $b_i < 0.6 \times 2^{i+4}$, then all honest users can connect to Tor with high probability.*

Proof. We first consider the execution of one of the $3 \log n$ repeats of Algorithm 1. For each user, the algorithm chooses a bridge independently and uniformly at random and assigns it to the user. Without loss of generality, assume the corrupt users are assigned bridges first.

For $k = 1, 2, ..., t_i$, let $\{X_k\}$ be a sequence of random variables each representing the bridge assigned to the k-th corrupt user. Also, let Y be a random variable corresponding to the number of *bad* bridges (i.e., the bridges that are assigned to at least one corrupt user) after all t_i corrupt users are assigned bridges. The sequence $\{Z_k = \mathrm{E}[Y|X_1, ..., X_k]\}$ defines a Doob martingale [?, Chapter 5], where $Z_0 = \mathrm{E}[Y]$. Since each corrupt user is assigned a fixed bridge with probability $1/d_i$, the probability that the bridge is assigned to at least one corrupt user is $1 - (1 - 1/d_i)^{t_i}$. By symmetry, this probability is the same for all bridges. Thus, by linearity of expectation,

$$\mathrm{E}[Y] = \left(1 - (1 - 1/d_i)^{t_i}\right) d_i < (1 - e^{-(t_i+1)/d_i}) d_i.$$

We know $t_i < 2^{i+4}$, because in each round $d_i = 2^{i+4}$ bridges are distributed and each corrupt user is assigned exactly one bridge; thus, each corrupt user can block at most one bridge. Hence,

$$\mathrm{E}[Y] < (1 - 1/e^{1+1/2^{i+4}}) d_i \leq (1 - 1/e^2) d_i \qquad (1)$$

Therefore, in expectation at most a constant fraction of the bridges become bad in each instance of the algorithm.

Since $|Z_{k+1} - Z_k| \leq 1$, $Z_0 = \mathrm{E}[Y]$, and $Z_{t_i} = Y$, by the Azuma-Hoeffding inequality [?, Theorem 5.2],

$$\Pr\left(Y > \mathrm{E}[Y] + \lambda\right) \leq e^{-2\lambda^2/t_i},$$

for any $\lambda > 0$. By setting $\lambda = \sqrt{d_i}$, we have

$$\Pr(Y > \mathrm{E}[Y] + \sqrt{d_i}) \leq e^{-2d_i/t_i} < 1/e^2. \qquad (2)$$

The last step holds since $t_i < d_i$. Therefore, with at most a constant probability, the actual number of bad bridges is larger than its expected value by at most $\sqrt{d_i}$. Therefore, the probability that an honest user is assigned a bad bridge is at most

$$\begin{aligned}\frac{\mathrm{E}[Y] + \sqrt{d_i}}{d_i} &< \frac{(1 - 1/e^{1+1/2^{i+4}}) d_i + \sqrt{d_i}}{d_i} \\ &= 1/e^{1+1/2^{i+4}} + 1/\sqrt{d_i}, \qquad (3)\end{aligned}$$

where the first step is achieved using (1).

Now, let $p_1 = \Pr(Y > \mathrm{E}[Y] + \sqrt{d_i})$, and let p_2 be the probability that a fixed honest user is assigned a bad bridge in a fixed instance and a fixed round. From (2) and (3), we have

$$p_1 < 1/e^2 \quad \text{and} \quad p_2 < 1/e^{1+1/2^{i+4}} + 1/\sqrt{d_i}.$$

Thus, the probability that a fixed user fails to receive a good bridge in a fixed instance and a fixed round is equal to $p_1 + (1 - p_1)p_2$, which is at most 0.6.

If the algorithm is repeated $\lceil 3 \log n \rceil$ times in parallel, then the probability that the user is assigned to only bad bridges in the last round is at most $0.6^{\lceil 3 \log n \rceil} \leq 1/n^2$. By a union bound, the probability that any of the n users is assigned a bad bridge in a round is at most $1/n$. Therefore, all honest users can connect to Tor with high probability. □

Algorithm 1 does not necessarily assign the same number of users to each bridge. However, in the following lemma, we show that each bridge is assigned to almost the same number of users as other bridges with high probability providing a reasonable level of load-balancing.

Algorithm 1 TorBricks – Basic Algorithm

Goal: Distributes a set of $O(t \log n)$ bridges among a set of users $\{u_1, ..., u_n\}$.

```
Run $3 \log n$ instances of this algorithm in parallel:
 1: Initialize parameters: $i \leftarrow 0$; $b_i \leftarrow 16$
 2: while true do
 3:     if $b_i \geq 0.6 \times 2^{i+4}$ then
 4:         $i \leftarrow i + 1$
 5:         $d_i \leftarrow 2^{i+4}$
 6:         if $d_i < \frac{n}{3 \log n}$ then
 7:             $\{B_1, ..., B_{d_i}\} \leftarrow d_i$ unblocked bridges
 8:             for each $j \in [n]$ do
 9:                 Pick $k \in [d_i]$ uniformly at random
10:                 Assign bridge $B_k$ to user $u_j$
11:             end for
12:         else
13:             Assign a unique bridge to every user
14:             break
15:         end if
16:     end if
17:     $b_i \leftarrow 0$
18:     for each $j \in [d_i]$ do
19:         if $B_j$ is not reachable then          ▷ Using [?, ?]
20:             $b_i \leftarrow b_i + 1$
21:         end if
22:     end for
23: end while
```

Lemma 2 (Bridge Load-Balancing). *Let X be a random variable representing the maximum number of users assigned to any bridge, Y be a random variable representing the minimum number of users assigned to any bridge, and $z = \Theta\left(\frac{\ln n}{\ln \ln n}\right)$. Then, we have*

$$\Pr\left(X \geq \mu z\right) \leq 2/n \quad \textit{and} \quad \Pr\left(Y \leq \mu z\right) \leq 2/n,$$

where $\mu = n/d_i$.

Proof. Each round of Algorithm 1 can be seen as the classic balls-and-bins process: n balls (users) are thrown independently and uniformly at random into d_i bins (bridges). It is well known that the distribution of the number of users assigned to a bridge is approximately Poisson with $\mu = n/d_i$ [**?**, Chapter 5].

Let X_j be the random variable corresponding to the number of users assigned to the j-th bridge, and let $\tilde{X}_j$ be the Poisson random variable approximating X_j. We have $\mu = \mathrm{E}[X_j] = \mathrm{E}[\tilde{X}_j] = n/d_i$. We use the following Chernoff bounds from [**?**, Chapter 5] for Poisson random variables:

$$\Pr(\tilde{X}_j \geq x) \leq e^{-\mu}(e\mu/x)^x, \text{ when } x > \mu \tag{4}$$

$$\Pr(\tilde{X}_j \leq x) \leq e^{-\mu}(e\mu/x)^x, \text{ when } x < \mu \tag{5}$$

Let $x = \mu y$, where $y = ez$. From (4), we have

$$\begin{aligned} \Pr(\tilde{X}_j \geq \mu y) &\leq \left(\frac{e^{y-1}}{y^y}\right)^\mu \\ &\leq \frac{e^{y-1}}{y^y} \\ &= \frac{1}{e}\left(\frac{1}{z^z}\right)^e < \frac{1}{n^2}. \end{aligned} \tag{6}$$

The second step is because $y^y > e^{y-1}$ (since $z > 1$) and $\mu > 1$. The last step is because $z = \Theta\left(\frac{\ln n}{\ln \ln n}\right)$ is the solution of $z^z = n$. To show this, we take log of both sides of $z^z = n$ twice, which yields

$$\ln z + \ln \ln z = \ln \ln n.$$

We have

$$\ln z \leq \ln z + \ln \ln z = \ln \ln n < 2 \ln z.$$

Since $z \ln z = \ln n$,

$$z/2 < \frac{\ln n}{\ln \ln n} \leq z.$$

Therefore, $z = \Theta\left(\frac{\ln n}{\ln \ln n}\right)$.

It is shown in [**?**, Corollary 5.11] that for any event that is monotone in the number of balls, if the event occurs with probability at most p in the Poisson approximation, then it occurs with probability at most $2p$ in the exact case. Since the maximum and minimum bridge loads are both monotonically increasing in the number of users, from (6) we have

$$\Pr(X_j \geq \mu y) \leq 2 \Pr(\tilde{X}_j \geq \mu y) < 2/n^2.$$

By applying a union bound over all bridges, the probability that the number of users assigned to any bridge will be more than μz is at most $2/n$. The bound on the minimum load can be shown using inequality (5) in a similar way. □

Lemma 3 (Latency). *By running Algorithm 1, all honest users can connect to Tor with high probability after at most $\lceil \log \lceil (t+1)/32 \rceil \rceil + 1$ iterations of the while loop.*

Proof. Let k denote the smallest number of rounds required until all users can connect to Tor with high probability. Intuitively, k is bounded, because the number of corrupt nodes, t, is bounded. In the following, we find k with respect to t.

Without loss of generality, we only consider one of the $3 \log n$ parallel instances of Steps 1–23 of Algorithm 1. The best strategy for the adversary is to maximize k, because this prevents the algorithm from succeeding soon. In each round i, this can be achieved by minimizing the number of bridges blocked (i.e., b_i), while ensuring the algorithm proceeds to the next round. However, the adversary has to block at least $0.6 \times 2^{i+4}$ bridges in each round to force the algorithm to proceed to the next round. Let ℓ be the smallest integer such that $2^\ell \geq t$. In round ℓ, the adversary has enough corrupt users to take the algorithm to round $\ell + 1$. However, in round $\ell + 1$, the adversary can block at most $2^\ell < 2^{\ell+1}$ bridges, which is insufficient for proceeding to round $\ell + 2$. Therefore, $\ell + 1$ is the last round and $k = \ell + 1$. Since $2^\ell \geq t$, and the algorithm starts by distributing 32 bridges,

$$k = \lceil \log \lceil (t+1)/32 \rceil \rceil + 1.$$

In other words, if the while loop runs for at least $\lceil \log \lceil (t+1)/32 \rceil \rceil + 1$ iterations, then with high probability all honest users can connect to Tor. □

LEMMA 4 (BRIDGE COST). *The total number of bridges used by Algorithm 1 is at most* $(10t + 96)\log n$.

PROOF. Consider one of the $3 \log n$ instances of Algorithm 1. The algorithm starts by distributing 32 bridges. In every round $i > 0$, the algorithm distributes a new bridge only to replace a bridge blocked in round $i - 1$. Thus, the total number of bridges used until round i, denoted by M_i, is equal to the number of bridges blocked until round i plus the number of new bridges distributed in round i, which we denote by a_i. Therefore,

$$M_i = a_i + \sum_{j=0}^{i-1} b_j. \tag{7}$$

In round i, the algorithm recruits $a_i \leq 2^{i+4}$ new bridges, because some of the bridges required for this round might be reused from previous rounds. Since in round i we have $b_i < 0.6 \times 2^{i+4}$,

$$M_i < 2^{i+4} + 0.6 \sum_{j=1}^{i-1} 2^{j+4} = 9.6(2^i - 2) + 2^{i+4}$$

From Lemma 3, it is sufficient to run the algorithm $k = \lceil \log \lceil (t+1)/32 \rceil \rceil + 1$ rounds. Therefore,

$$M_k < 9.6(2^k - 2) + 2^{k+4} \leq 3.2t + 32.$$

Since the algorithm is repeated $3 \log n$ times, the total number of bridges used by the algorithm is at most $(10t + 96)\log n$. □

3.1.2 Handling Client Churn

Algorithm 1 can only distribute bridges among a fixed set of users. A more realistic scenario is when users join or leave the algorithm frequently. One way to handle this is to add the new users to the algorithm from the next round (i.e., an increment of i). This, however, introduces two technical challenges:

1. The number of corrupt users is unknown, and hence the adversary can arbitrarily delay the next round, causing a denial of service attack; and

2. Our proof of robustness (Lemma 1) does not necessarily hold if n is changed, because we repeat the algorithm $3 \log n$ times to ensure it succeeds with high probability.

To handle these challenges, we add the following steps to Algorithm 1:

1. Each time a user wants to join the system, assign him to $3 \log n$ random bridges from the set of bridges recruited in the last round (i.e., the last time i was incremented);

2. If the total number of users, n, is doubled since the last round, recruit $3 \times 2^{i+4}$ unblocked bridges and assign 3 of them randomly to each user.

The first step guarantees that the new users are always assigned bridges once they join the system. The second step ensures that the number of parallel instances always remains $3 \log n$ even if n is changed. This is because $\log n$ is increased by one when n is doubled. Therefore, each existing user must receive 3 new bridges so that the proof of Lemma 1 holds in the setting with churn. Our previous lemmas hold if users leave the system; thus, we only need to update n once they leave.

Since distributing new bridges among existing users is done only after the number of users is doubled, the latency is increased by at most a $\log n$ term, where n is the largest number of users in the system during a complete run of the algorithm.

3.2 Privacy-Preserving Algorithm

We now adapt Algorithm 1 to the multiple distributor setting. Our goal is to run a protocol jointly among multiple distributors to keep user-bridge assignments hidden from each distributor and any coalition of up to a 1/3 fraction of them. We assume that a sufficient number of bridges have already registered their email addresses in the system so that in each round the protocol can ask some of them to provide their IP addresses to the system to be distributed by the protocol.

We first construct a *leader-based protocol*, where an honest-but-curious distributor called the *leader* locally runs Algorithm 1 over anonymous bridge addresses. The leader then sends anonymous user-bridge assignments to other distributors who can collectively "open" the assignments for the users.

Next, we construct a fully decentralized protocol, where a group of m distributors collectively compute the bridge distribution functionality while resisting malicious fault from up to a $\lfloor m/3 \rfloor$ fraction of the distributors. Malicious distributors not only may share information with other malicious entities but also can deviate from our protocol in any arbitrary manner, e.g., by sending invalid messages or remaining silent.

Both of these protocols rely on a secret sharing scheme for the bridges to share their IP addresses with the group of distributors. Before proceeding to our protocols, we briefly describe the secret sharing scheme used in our protocol.

Secret Sharing. A *secret sharing* protocol allows a party (called *the dealer*) to share a secret among m parties such that any set of τ or less parties cannot gain any information about the secret, but any set of at least $\tau + 1$ parties can reconstruct it. Shamir [?] proposed a secret sharing scheme, where the dealer shares a secret s among m parties by choosing a random polynomial $f(x)$ of degree τ such that $f(0) = s$. For all $j \in [m]$, the dealer sends $f(j)$ to the j-th party. Since at least $\tau + 1$ points are required to reconstruct $f(x)$, no coalition of τ or less parties can reconstruct s. The reconstruction algorithm requires a Reed-Solomon decoding algorithm [?] to correct up to 1/3 invalid shares sent by dishonest distributors. In our protocol, we use the error correcting algorithm of Berlekamp and Welch [?].

3.2.1 Leader-Based Protocol

Similar to Algorithm 1, the leader-based protocol also proceeds in rounds. In each round i, the leader requests a group of at most d_i bridges to secret-share their IP addresses among all distributors (including the leader) using Shamir's scheme [**?**].

Let $(B_1, ...B_{d_i})$ denote the sequence of shares the leader receives once the bridges finish the secret sharing protocol. The leader runs Algorithm 1 locally to assign B_j's to the users randomly, for all $j \in [d_i]$. Then, the leader broadcasts the pair (u_k, I_k) to all distributors, where I_k is the set of indices of bridges assigned to user u_k, for all $k \in [1, ..., n]$.

Each distributor then sends its shares of bridge addresses to the appropriate user with respect to the assignment information received from the leader. Finally, each user is able to reconstruct the bridge addresses assigned to him, because at least a 2/3 fraction of the distributors are honest and have correctly sent their shares to the user.

3.2.2 Fully Decentralized Protocol

In each round, Algorithm 1 picks one of the d_i bridges uniformly at random. For each user u, if the group of distributors described in the leader-based protocol can collectively agree on a random number $k \in [d_i]$, then each of them can individually run Algorithm 1 to assign u to the bridge corresponding to k.

Assuming each distributor holds a share of every bridge address (similar to the leader-based protocol), he can then send his share to u, allowing the user to privately reconstruct the bridge address even if at most a 1/3 fraction of the shares are invalid.

Distributed Random Generation. We use a well known commit-and-reveal technique for distributed random generation [**?**, **?**]. This protocol has at most four rounds of communication and can run efficiently among a small number of distributors to generate unbiased random numbers even if up to a $m/3$ distributors play maliciously.

Let $D_1, ..., D_m$ denote the distributors. The DRG protocol starts by each distributor D_j choosing a uniform random number r_j locally and then publishing a commitment c_j to it. Once all commitments have been received, the distributors reveal their random numbers and verify the commitments. Finally, each distributor computes $r = \sum_{k=1}^{m} r_j$. In TorBricks, we use the simple commitment mechanism described in [**?**].

Although this protocol can generate unbiased random numbers, it is vulnerable to *equivocation attacks*[3]: A dishonest distributor can send different random values to different distributors while the corresponding commitments are correct and check out. We prevent this by asking the distributors to participate in a Byzantine agreement protocol to agree on the random value r at the end of the protocol. In TorBricks, we use the small scale Byzantine agreement protocol of Castro and Liskov [**?**] known as BFT. This protocol is efficient for small numbers of participants (about 10) and can tolerate faults from up to a third fraction of the participants. If any of the distributors equivocates, then the BFT protocol fails, and the DRG protocol restarts.

Another vulnerability is *denial of service attacks*: A dishonest distributor can bias the random number by choosing whether or not to open his commitment; using this he can repeatedly cause the protocol to abort and restart until the resulting value is what he desires. In both equivocation and denial of service attacks, the cheating distributors can be detected quickly using administrative mechanisms, especially since the number of distributors is small in our model. Therefore, we believe these attacks do not offer much gain to the adversary. We finish this section by analyzing the communication complexity of our protocol.

LEMMA 5 (COMMUNICATION COMPLEXITY). *In each round of TorBricks, each user sends/receives at most m messages and each distributor sends/receives $O(m^2 + n)$ messages. Each message has length $O(\log n)$ bits.*

PROOF. In the single distributor model, the distributor sends one message to each client per round. Each message contains a list of $3 \log n$ bridge addresses, therefore it has length $O(\log n)$ bits.

In the multiple distributors models (leader-based and fully-decentralized), each distributor sends one message to each client per round. Therefore, each client receives m messages in each round. Since each message contains a list of $3 \log n$ secret-shared values (each corresponding to a bridge address), each message is of size $O(\log n)$ bits.

In the multiple distributor models, each distributor receives a secret-shared value from each bridge. Since the total number of bridges used by the protocol is $O(t \log n)$, each distributor receives $O(t \log n)$ field elements in all rounds from the bridges.

In the leader-based model, the leader sends to every distributor one message each containing a list of $3 \log n$ user-bridge information. Therefore, the leader sends a total of m messages each of size $O(\log n)$ bits in each round. Each distributor in this model sends to each client one message each containing $3 \log n$ secret-shared values, thus each distributor sends/receives

$$O\left(m + n + \frac{t \log n}{\log t}\right) = O(m + n)$$

messages of size $O(\log n)$ bits in each round.

In the fully decentralized model, each distributor participates in a run of the random generation protocol per round. This protocol consists of one secret sharing round transmitting m field elements per distributor, and one run of the BFT protocol, which sends $O(m^2)$ field elements per distributor. Finally, each distributor sends to each client one message containing a list of $3 \log n$ secret-shared values. Thus, each distributor sends/receives

$$O\left(m^2 + n + \frac{t \log n}{\log t}\right) = O(m^2 + n)$$

messages each of size $O(\log n)$ bits in each round.

□

4. EVALUATION

We implemented a proof-of-concept prototype and tested it in a simulated environment under various adversarial behavior. The prototype is written in C# using .NET Framework 4.5. We ran the simulations on an Intel Core i5-4250U

[3] In [**?**], these attack are referred to as *partition attacks*.

1.3GHz machine with 4GB of RAM running Windows 10 Pro. We set the parameters of TorBricks in such a way that we ensure it fails with probability at most 10^{-4}.

We consider three blocking strategies for the adversary: *prudent*, *aggressive*, and *stochastic*. A prudent adversary blocks the minimum number of bridges in each round such that the algorithm is forced to go to the next round. An aggressive adversary blocks immediately all of the bridges he learns from the corrupt users. Finally, a stochastic adversary blocks each bridge he receives with some fixed probability.

To evaluate the performance of TorBricks, we calculate five measures of performance in two experiments. These measures are:

1. Thirsty Users: Number of users who do not have any unblocked bridge in the current round.

2. Bridges Distributed: Number of bridges distributed in the current round (d_i).

3. Bridges Blocked: Number of bridges blocked in the current round (b_i).

4. Bridges Used: Total number of unique bridges distributed by the algorithm until this round (M_i).

5. Latency: Number of round until all users receive at least one unblocked bridge.

In the first experiment (shown in Figure 5), we run the algorithm in the basic algorithm for $n = 65536$ and $t = 180$, and calculate Measures 1–4 at the end of each round after running the algorithm over ten samples for a fixed set of parameters. The experiment was run with the three different adversarial strategies. For the stochastic blocking, the adversary blocks each bridge with probability 0.95. In the second experiment (shown in Figure 6), we run the algorithm using a single distributor for $n = 1024$ and calculate Measures 4 and 5 by varying t between 0 and 1023.

Our results indicate that TorBricks incurs a small cost when there is small or no corruption in the network. Moreover, the algorithm scales well with the number of corruptions and can quickly adapt to the adversary's behavior. These all support our claim that TorBricks can be used for practical bridge distribution with certain access guarantee for all users.

5. CONCLUSION

We described TorBricks, a bridge distribution system that allows all honest users to connect to Tor in the presence of an adversary corrupting an unknown number of users. Our algorithm can adaptively increase the number of bridges according to the behavior of the adversary and use near-optimal number bridges. We also modified our algorithm slightly to handle user churn (join/leave) by adding small (constant) amortized latency.

We also described a protocol for privacy-preserving bridge distribution by running the distribution algorithm obliviously among a group of distributors. We showed that the resulting protocol not only can protect the privacy of user-bridge assignments from any coalition of up to a 1/3 fraction of the distributors but also can tolerate malicious attacks from a 1/3 fraction of the distributors. We finally evaluated a prototype of our protocol in different simulated scenarios to show that TorBricks can be used efficiently in practice.

Although TorBricks represents a step towards robust and privacy-preserving bridge distribution, many challenges remain for future work. For example, the current algorithm uses a relatively large number of bridges when the number of corrupt users is large. Is it possible to make the bridge cost sublinear in t with practical constant terms?

An exciting direction for answering this question to use inexpensive honeypot bridges for detecting and blacklisting corrupt users. This, however, requires a mechanism such as CAPTCHA for preventing the adversary from distinguishing real bridges from the fake ones. Moreover, a colluding adversary may be able to compare bridges assigned to its corrupt users to detect honeypots.

To better explore the possibility of achieving a sublinear bridge cost, one may consider finding lower bounds for different scenarios. For example, when each user is assigned at least one bridge, it seems impossible to achieve a sublinear bridge cost unless some of the bridges are fake, or we only distribute real bridges in random-chosen rounds. What is the lower bound for the number of rounds in these scenarios?

Another interesting open problem is to examine if our current notion of robustness is overkill for practice. For example, is it possible to significantly reduce our costs by guaranteeing access for all but a constant number of users?

6. REFERENCES

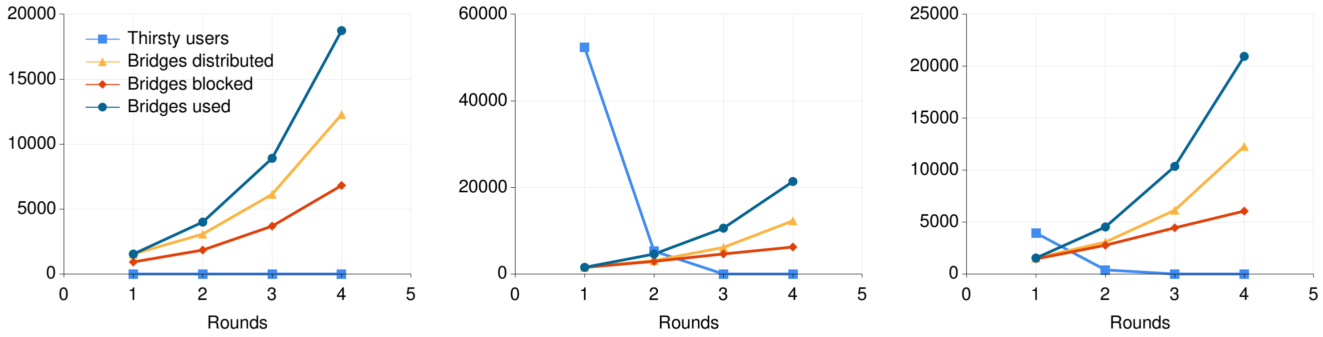

Figure 5: Simulation results for $n = 65536$ and $t = 180$ with prudent (left), aggressive (middle), and stochastic (right) adversary.

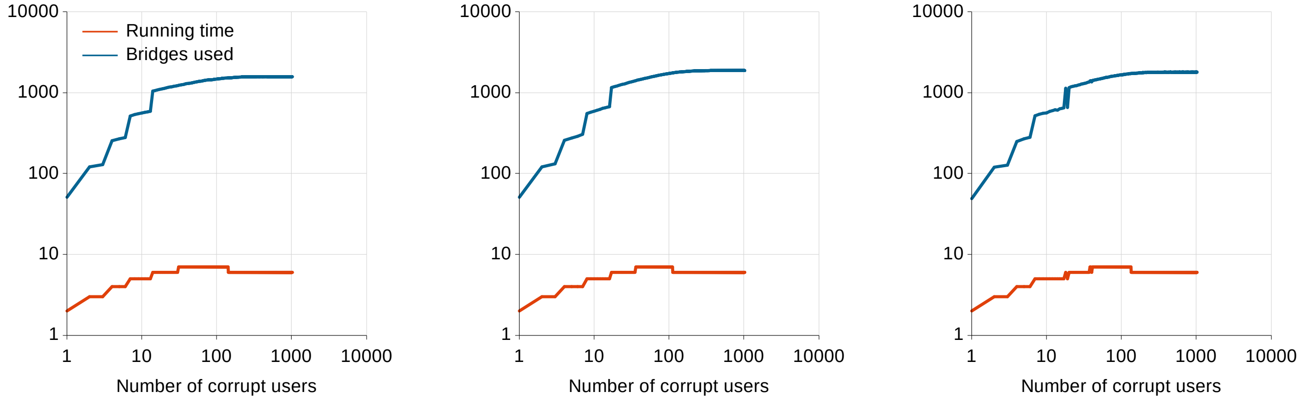

Figure 6: Simulation results for $n = 1024$ and variable number of corrupt users with prudent (left), aggressive (middle), and stochastic (right) adversary.