Spread the Word:
Blocking-Resistant Tor Bridge Distribution

Jedidiah R. Crandall, Jared C. Saia, and Mahdi Zamani
Department of Computer Science, University of New Mexico
{crandall, saia, zamani}@cs.unm.edu

October 4, 2015

Abstract

Tor is a popular anonymity network that relays Internet traffic through a world-wide network of volunteer nodes. These relays are publicly-known and hence may be blocked by censorship systems. To make blocking harder, Tor uses the notion of bridges; volunteer nodes across the globe that direct user traffic to the relays, but there is no complete public list of them. An important challenge with the current bridge distribution mechanism of Tor is that malicious users can still obtain information about a large number of bridges and block them.

In this paper, we describe a randomized bridge distribution algorithm, where all (honest) users are guaranteed to be able to connect to Tor in the presence of an adversary corrupting an unknown number of users $t < n$, where $n$ is the total number of users. Our algorithm adaptively increases the number of bridges according to the behavior of the adversary and requires $\tilde{O}(t)$ bridges. We show that, using our algorithm, the number of times a user fails to connect to Tor via bridges is bounded by $O(\log t)$ with high probability.

1 Introduction

Today, mass surveillance and censorship severely threaten democracy and freedom of speech. Governments around the world control the Internet to protect their domestic political, social, financial, and security interests. This makes anonymity a crucial tool for preserving privacy of individuals in cyberspace. Tor [DMS04] is the most popular anonymous communication network with more than 2.5 million users per day [Tor15a]. Tor relays the Internet traffic via more than 6,000 volunteer nodes called relays spread across the world. Tor users can connect to the network and have their Internet data routed through the network before reaching any server, thus the servers are not able to distinguish between Tor users or locate them.

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 – volunteer relay nodes that are not all listed in Tor’s public directory [DM06]. 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 be all learned by censorship authorities. Tor users behind censorship firewalls must find bridges through email, uncensored web sites, etc.

Although currently the bridges are distributed among the users based on different strategies such as CAPTCHA-enabled email-based distribution and IP-based distribution [DM06], studies indicate that censors
are using sophisticated mechanisms along with a large coalition of corrupt users (scanners) to obtain and block many bridge addresses rendering Tor unavailable for many users [Din11b, WL12, EFW*15].

In this paper, we study the problem of bridge distribution in Tor, where a set of bridges is distributed among $n$ users, $t$ of whom are controlled by a censor, in such a way that all honest users can access a bridge that is not blocked by the censor. A solution to this problem would allow us to make the Tor network provably available for all users. Unfortunately, state of the art techniques for bridge distribution either cannot guarantee that all honest users can access Tor [WLBH13, MML12] or only work when $t$ is known in advance [Mah10].

In contrast, we describe an algorithm that ensures Tor is available to all honest users with high probability without requiring any a priori knowledge about $t$. It is often hard in practice to estimate the number of corrupt users due to the sophisticated nature of Internet censorship in many countries such as China [Oni12, EFW*15]. Moreover, censors can easily implement strategies to prevent the circumvention mechanism from correctly estimating the number of corrupt users.

Inspired by the resource competitive analysis approach of Gilbert et al. [GSKY12] and Bender et al. [BFM+15], our algorithms provide the following guarantee: if the adversary pays $t$ amount of resource cost, then the resource cost of our algorithms is some function of $t$. This allows us to achieve near-optimal resource costs with strong robustness to disruptions caused by the censor.

Our main strategy for preventing the censor from blocking a large fraction of the bridges is to use randomization. This is because the colluding censor cannot predict the behavior of the randomized process in distributing a set of bridges to the users, and thus it cannot arrange its corrupt users in such a way that prevents some of the honest users from connecting to the Tor network. Moreover, our algorithm can adaptively adjust the number of bridges required based on the number of bridges compromised so far and guarantees that the number of rounds until all honest users can connect to Tor is bounded by $O(\log t)$. Our algorithms can be run independently from Tor; the Tor network remains intact to focus on its main purpose of providing anonymity.

In Section 1.1, we describe our communication and adversarial model. In Section 1.2, we state our main theorem. We review the previous results and 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 summarize the paper and state a few open problem in Section 4.

### 1.1 Our Model

Consider $n$ users (clients) connected to the Internet via an ISP inside a censorship territory and a trusted server called the bridge distributor (or simply the distributor) connected to the Internet via an ISP outside the territory. The distributor has information about a set of $m$ Tor bridges that are also located outside the territory. Each user wants to obtain information about a subset of the bridges at least one of which can be used to connect to Tor. The user sends its request to the distributor using a rate-limited channel\(^1\) such as email and the distributor sends a set of bridges back to the user via the same method of contact.

Among the users, there are $t < n$ corrupt users controlled by an adaptive adversary (the censor) who can choose the set of corrupt users at the beginning of each round of communication, but is limited to corrupting at most $t$ distinct users combined in all rounds. We refer to the other $n - t$ users as honest users. Each corrupt user is willing to obtain information about the bridges in order to block them, but does not have to block a bridge as soon as it finds the bridge; it is allowed to strategically (perhaps by colluding with other corrupt users) to decide when to block a bridge. Moreover, we assume our algorithms do not have any knowledge about $t$ before they begin. Figure 1 depicts our model.

\(^1\)A vital communication mechanism such as Gmail that the censor cannot block due to major economic, political, etc consequences, but is not suitable for interactive communication over the Internet such as web surfing.
1.2 Our Results

We prove the following main theorem in Section 3.1.1.

**Theorem 1.** There exists a bridge distribution algorithm that guarantees the following properties with probability $1 - 1/n^k$, for any $k > 0$:

1. All honest users can connect to Tor after $\lceil \log t \rceil + 2$ rounds of communication with the distributor.
2. The total number of bridges required is $O(t)$.

The best known algorithm for bridge distribution [Mah10] only works when the number of corrupt users, $t$, is known in advance and requires at most $k (1 + \lceil \log (n/k) \rceil)$ bridges. In contrast, our algorithms not only do not require any prior knowledge about $t$ but also use fewer bridges.

2 Related Work

The bridge distribution problem can be seen as a special case of the proxy distribution problem, where a set of unblocked servers (proxies) outside the censorship territory are distributed among the users inside the territory. These proxies are used to relay Internet traffic to blocked websites. The proxy distribution problem has been studied by several previous work.

Feamster *et al.* [FBW+03] propose a proxy distribution algorithm that requires every user to solve a cryptographic puzzle to discover a proxy. This way, the algorithm prevents the corrupt users from learning a large number of proxies. Unfortunately, empirical results of [FBW+03] show that a computationally powerful censor can easily block a very large fraction of the proxies.

The Kaleidoscope system of Sovran *et al.* [SLL08] 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 and cannot guarantee its users’ access to Tor.

McCoy *et al.* [MML12] 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.
Even if bridges are distributed carefully among the users, censors can still block access to the Tor network via deep packet inspection (DPI). The Tor Project has developed a variety of tools under the name pluggable transports [Tor15b] that can be used to obfuscate the traffic transmitted between the client and the bridge. This makes it hard for the censor (who monitors the traffic) to distinguish between the legitimate-looking obfuscated traffic and the actual Tor traffic. Although pluggable transports are necessary for preventing bridge blocking via DPI, they cannot prevent blocking via colluding corrupt users. Moreover, recently Wang et al. [WDA+15] showed that current obfuscation mechanisms used in Tor can be reliably detected by censors with sufficiently low false-positive rates. In our model, we assume our algorithm runs in parallel with a reliable pluggable transport tool that can prevent DPI blocking.

Mahdian [Mah10] 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 \( k (1 + \lceil \log(n/k) \rceil) \) bridges, and his algorithm for small known \( t \) uses \( \Omega(k^2 \log n / \log \log n) \) bridges.

Wang et al. [WLBH13] 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 achievingunlinkability of transactions.

Our algorithms rely on a technique for testing reachability of bridges from outside the censored territory. Dingledine [Din11a] and Ensafi et al. [EKAC14] describe active scanning mechanisms for testing availability of bridges. The details of these methods and their current challenges are out of scope of our paper.

3 Our Algorithms

In this section, we first construct a bridge distribution algorithm that adaptively increases the number of bridges used with respect to the number of bridges blocked. In Section 3.1.1, we prove the desired properties of this algorithm. In Section 3.2, we improve our result by constructing a detection-and-eviction mechanism, where blocking parties are gradually detected and removed from the algorithm. We show properties of this algorithm in Section 3.2.1. Before proceeding to our algorithms, we define the standard terms and notation used throughout our paper.

Notation. We say an event occurs with high probability, if it occurs with probability \( 1 - o(1) \). We denote the set of integers \( \{1, \ldots, n\} \) by \([n]\). We denote a set of \( n \) users participating in our algorithms by \([u_1, \ldots, u_n]\). We say a bridge is blocked when the censor has restricted users’ access to this bridge. We refer to the remaining bridges as unblocked bridges.

3.1 Simple Bridge Distribution Algorithm

Our simple method is a Monte Carlo algorithm that proceeds in rounds (see Algorithm 1): in each round, the algorithm randomly distributes a set of bridges among the users and proceeds to the next round once the number of blocked bridges exceeds a threshold that is increased exponentially in each round. In every round, we increase the number of fresh bridges with respect to the number of bridges blocked so far. For

\[ 2 \text{ For large values of } t \text{ (e.g., } t = cn \text{ for } c \in (0, 1) \text{) the trivial lower bound of } \Omega(t) \text{ is better than } \Omega\left(\frac{t \log(n(t))}{\log t + \log \log n}\right) \text{ of [Mah10].} \]

\[ 3 \text{ We may alternatively say this event fails with } o(1) \text{ probability.} \]
some \( c > 1 \), we run \( \lceil c \log n \rceil \) instances of this algorithm in parallel. In Lemma 1, we show that each instance guarantees that all users can connect to Tor with constant probability. Therefore, running \( \lceil c \log n \rceil \) instances in parallel guarantees that all users can connect to Tor with high probability.

**Algorithm 1** Simple Bridge Distribution Scheme

For some \( c > 1 \), run \( \lceil c \log n \rceil \) instances of the following algorithm in parallel:

1. Initialize parameters: \( i \leftarrow 0; b_i \leftarrow 1; U \) a set of users \( \{u_1, ..., u_n\} \)
2. **while** true **do**
   3. \( \text{if } b_i \geq 2^i \text{ then} \)
   4. \( i \leftarrow i + 1 \) \( \triangleright \) Proceed to the next round
   5. \( B_i = \text{Distribute}(U, 2^{i+1}) \) \( \triangleright \) Distribute \( 2^{i+1} \) unblocked bridges
   6. **end if**
   7. \( b_i \leftarrow \text{number of bridges in } B_i \text{ that are blocked} \)
3. **end while**

We refer to a single execution of the while loop in Algorithm 1 as an *iteration*. We refer to each increment of the variable \( i \) (in line 4) as a *round*. Note that several iterations may correspond to the same round depending on the condition in line 3 of the algorithm. For simplicity, we assume the \( \lceil c \log n \rceil \) instances of Algorithm 1 run *synchronously* meaning that they start and finish each iteration at the same time.\(^4\) We also assume that each iteration runs *atomically* meaning that the users are allowed to use (or block) the bridges assigned to them (in any round) only at the end of each iteration.

Algorithm 2 implements the function Distribute. Let \( w \) be the number of bridges distributed in each round. In each run of Distribute, an \( \frac{n}{w} \times w \) matrix is created and each user in \( U \) is randomly assigned to one of the elements of the matrix such that all users appear in the matrix and each user appears exactly once. The random assignment of users is done using a random permutation \( \pi \) that maps every integer between 1 and \( n \) (corresponding to every user index).

Next, the algorithm recruits a set of \( w \) fresh (unblocked) bridges and assigns a unique bridge to all users in each column of the matrix. To improve the efficiency of our algorithm in practice, we assume the bridges that were recruited in previous rounds and remain unblocked are reused for distribution in the next round.

**Algorithm 2** Randomized Bridge Distribution

**Goal:** A sequence of \( w_i \) bridges is randomly distributed among a set of \( n \) users \( U = \{u_1, ..., u_n\} \).

1. **function** Distribute\((U, w_i)\)
2. Initialize parameter: \( n \leftarrow |U|; B_i \) a sequence of \( w_i \) unblocked bridges
3. Define a matrix \( M = \left[u \left( i+(j-1)\frac{w_i}{w} \right) \right]_{i=1}^{n} \times_{j=1}^{w_i} \) such that \( \pi : [n] \rightarrow [n] \) is a random permutation
4. **for all** \( j \in [w_i] \) **do**
   5. Assign \( B_i[j] \) to all users in the \( j \)-th column of \( M \)
5. **end for**
6. **return** \( B_i \)
7. **end function**

\(^4\) Since the distributor runs the instances locally, it is not hard to guarantee that they run synchronously.
Figure 2: Matrices generated by Algorithm 2 in round $i$

Figure 2 shows the matrix generated in each execution of Algorithm 2. In this figure, $\pi : [n] \rightarrow [n]$ refers to the random permutation generated in line 3 of the algorithm, and $B$ refers to the sequence of $w$ bridges being assigned to the users in the current run of the algorithm.

### 3.1.1 Proof of Algorithm 1

We now prove the properties of Algorithm 1. For simplicity, we assume a user can connect to Tor in an iteration if and only if at least one unblocked bridge is assigned to it. We say a corrupt user is active in round $i$ if it has blocked at least one bridge in round $i$. We now define the following notation used in our proof:

- $b_i$: the number of bridges blocked in round $i$.
- $w_i$: the number of columns of matrix $M$ generated by Algorithm 2 in round $i$.
- $t_i$: the number of corrupt users, where each has blocked at least one unblocked bridge in round $i$.
- $B_i$: the sequence of bridges distributed in round $i$.

**Lemma 1.** In the $i$-th round of Algorithm 1, if $b_i < 2^i$, then all honest users can connect to Tor with high probability.

**Proof.** Since the $n$ users are arranged uniformly and independently at random in $M$, and there are $t_i$ active corrupt users among them, the probability that for a given user $u$ the column that $u$ appears in contains at least one corrupt user is at most

$$1 - \left(\frac{n-t_i}{n}\right)^{n/w_i} = 1 - \left(1 - \frac{t_i}{n}\right)^{n/w_i} \leq \frac{t_i}{w_i}.$$  

The last inequality is correct based on the Bernoulli’s inequality when $t_i \leq w_i$. If $b_i < 2^i$, then $t_i < 2^i$ because in each round each corrupt user appears in exactly one column in $M$, and thus can block at most one bridge. Since in each round $w_i = 2^{i+1} \geq 2t_i$, this probability becomes a constant $\leq 1/2$.

Since $\lceil c \log n \rceil$ instances of Algorithm 1 run in parallel, the probability that every column that $u$ appears in among all matrices is “bad” (i.e., has at least one active corrupt user) is at most $(1/2)^{c \log n} = 1/n^c$. By applying the union bound, the probability that any of the $n$ users fails to sit in a “good” column (i.e., a column with no active corrupt user) is at most $1/n^{c-1}$. Therefore for any $c > 1$, all honest users can connect to Tor with high probability.

\[\Box\]
Lemma 2. By running Algorithm 1, all honest users can connect to Tor with high probability after at most \([\log t] + 2\) iterations.

Proof. Let \(k\) denote the minimum number of rounds needed until all users can connect to Tor with high probability. Intuitively, \(k\) is bounded, because \(t\) (the adversary’s budget) is bounded, and so there exists \(i \geq k\) such that \(b_i < 2^i\) and based on Lemma 1 all users can connect to Tor with high probability. In the following, we find an upper bound for \(k\).

The best strategy for the adversary is to maximize \(k\), because this prevents our 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. The adversary can do this by blocking only half of the \(2^{i+1}\) bridges distributed in each round and memorizing the rest \(2^i\) bridge addresses to be blocked in future rounds, where \(t < 2^i\). Let \(\ell\) be the smallest integer such that \(t \leq 2^\ell\). Until round \(\ell\), the adversary can memorize at most

\[
\sum_{j=1}^{\ell-1} 2^j = 2^\ell - 2
\]

bridge addresses. In round \(\ell\), no more bridges can be memorized, because the adversary has to block all of the \(t \leq 2^\ell\) bridges it has learned in this round to force the algorithm to proceed to round \(\ell + 1\). However in round \(\ell + 1\), the adversary can block at most \(2^\ell - 2 + 2^\ell = 2^{\ell+1} - 2 < 2^{\ell+1}\) bridges which is insufficient for proceeding to round \(\ell + 2\). Therefore, \(\ell + 1\) is the last round and \(k \leq \ell + 2\). Since \(t \leq 2^\ell\), \(k \leq [\log t] + 2\). In other words, if the algorithm is run for at least \([\log t] + 2\) iterations, then with high probability all honest users can connect to Tor.

Lemma 3. The total number of bridges used by Algorithm 1 is \(O(t \log n)\).

Proof. In every round \(i > 1\), the algorithm distributes a new bridge only to replace a bridge blocked in round \(i - 1\) (as in line 2 of Algorithm 2). Thus, the total number of bridges used until round \(i\) (denoted by \(N_i\)) 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 \(a_i\). We have

\[
N_i = a_i + \sum_{j=1}^{i-1} b_j.
\]

In the \(i\)-th round, Algorithm 1 recruits \(a_i \leq 2^{i+1}\) new bridges because some of the bridges required for this
round may be reused from previous rounds. Since in the $i$-th round, $b_i < 2^i$, we have

$$N_i < 2^{i+1} + \sum_{j=1}^{i-1} 2^j = 2^{i+1} + 2^i - 2 = 3 \cdot 2^i - 2$$

From Lemma 2, it is sufficient to run the algorithm for $\lceil \log t \rceil + 2$ rounds. Therefore, the number of bridges used by each instance of the algorithm is

$$N_i < 3 \cdot 2^{\lceil \log t \rceil + 2} - 2 < 24t - 2,$$

and the total number of bridges used by the algorithm is $(24t - 2)c \log n = O(t \log n)$. □

### 3.2 Detection Scheme

In this section, we describe a technique for detecting and blacklisting corrupt users which allows us to reduce the number of bridges used by the algorithm from $O(t \log n)$ to $O(t)$.

**Algorithm 3 Bridge Distribution with Detection Scheme and Fixed $n$**

1: Initialize parameters: $c > 0; i ← 1; U ←$ a set of users $\{u_1, ..., u_n\}$
2: Distribute$(2, c, U)$ ▷ Distribute $2c \log n$ real bridges
3: DistributeFakes$(c, U)$ ▷ Distribute $nc \log n$ fake bridges
4: while true do
5:     $b_i ←$ number of real bridges blocked so far ▷ Using [?] 
6:     From $U$, delete every user whom any of the fake bridges assigned to it is blocked
7:     if $b_i \geq i2^i c \log n$ then
8:         $i ← i + 1$ ▷ Proceed to the next round
9:     Distribute$(2^i, c, U)$ ▷ Distribute $2^i c \log n$ real bridges
10:    DistributeFakes$(c, U)$ ▷ Distribute $nc \log n$ fake bridges
11: end if
12: end while

**Algorithm 4 Fake Bridge Distribution**

Goal: A set of $nc \log n$ fake bridges is distributed among a set of $n$ users $U = \{u_1, ..., u_n\}$.

1: function DistributeFakes$(U)$
2:     Initialize parameter: $n ← |U|; F ←$ a list of $cn \log n$ unblocked fake bridges
3:     for all $j \in [n]$ do ▷ Assign $c \log n$ fake bridges to each user
4:         Assign $F[(j-1)c \log n + 1], ..., F[jc \log n]$ to $u_j$
5:     end for
6: end function

### 3.2.1 Proof of Algorithm 3

CORRECT PROOFS OF THIS SECTION USING PAGES 47-50 OF THE NOTEPAD
3.3 Sublinear Bridge Cost

In Algorithm 3, we assume each iteration is either successful or unsuccessful. Then, we show that \(O(t \log t \log n)\) bridges are needed to ensure that we get at least one successful iteration among \(O(\log t)\) iterations. Unfortunately, this algorithm and the corresponding analysis do not consider the number of honest users whom have been assigned at least one unblocked bridge even in unsuccessful iterations. In this section, we introduce a slightly modified model, where in the beginning of the algorithm, each user is holding one message to send to Tor. The goal is to guarantee that at the end of the algorithm, each user is given at least one chance to send its message to Tor. In other words, we guarantee that each user is given at least one unblocked bridge before the algorithm terminates.

Algorithm 5 Bridge Distribution Scheme

1: Initialize parameters: \(c > 0\); \(i \leftarrow 1\); \(U \leftarrow\) a set of users \(\{u_1, ..., u_{n_i}\}\); \(n_i \leftarrow |U|\)
2: Distribute\((2, c, U)\) \(\Leftrightarrow\) Distribute \(2c \log n\) unblocked bridges
3: while \(n_i > 0\) do
4: \(b_i \leftarrow\) number of bridges blocked so far \(\Leftrightarrow\) Using [?]\n5: if \(b_i \geq i2^i c \log n_i\) then
6: \(i \leftarrow i + 1\) \(\Leftrightarrow\) Proceed to the next round
7: for all \(k \in [c \log n]\) do
8: DistributeSingles\((2^i, c, U)\) \(\Leftrightarrow\) Distribute \(2^i c \log n\) unblocked bridges
9: end for
10: end if
11: From \(U\), remove the users that have successfully sent their message
12: \(n_i \leftarrow |U|\) \(\Leftrightarrow\) Update the number of users
13: end while

3.4 Proof of Algorithm 5

SEE PAGES 55-58 OF THE NOTEPAD

Lemma 4. Algorithm 5 terminates after \(O()\) iterations.

Proof. TBW. □

Lemma 5. The total number of bridges used in Algorithm 5 is \(O()\).

Proof. TBW □

Lemma 6. Each user in Algorithm 5 is assigned \(O()\) bridges.

Proof. TBW □

4 Conclusion

TBD
Algorithm 6 Randomized Real/Fake Bridge Distribution

\textbf{Goal:} A set of $w$ real bridges and $n$ fake bridges is randomly distributed among a set of $n$ users $U = \{u_1, ..., u_n\}$.

1: \textbf{function} DistributeSingles$(w, c, U)$
2: \hspace{1em} Initialize parameter: $n \leftarrow |U|$; $B \leftarrow$ a list of $wc \log n$ unblocked bridges
3: \hspace{1em} Define a matrix $M$ with $w$ columns and $\frac{n}{w}$ rows such that for all $i \in \lfloor \frac{n}{w} \rfloor$ and $j \in [w]$, we have $M_{i,j} = [u_{j+(i-1)w}]$.
4: \hspace{1em} $r \leftarrow$ a sequence of $\Theta(n \log n)$ bits chosen uniformly and independently at random
5: \hspace{1em} $M \leftarrow \text{Shuffle}(r, M)$
6: \hspace{1em} \textbf{for all} $j \in [w]$ \textbf{do}
7: \hspace{2em} \textbf{for all} $i \in \lfloor \frac{n}{w} \rfloor$ \textbf{do}
8: \hspace{3em} $x \leftarrow$ a value chosen randomly from $[0, 1]$
9: \hspace{3em} \textbf{if} $x \leq p$ \textbf{then}
10: \hspace{4em} Assign $B[j + (k-1)w]$ to the user in $M_{i,j}$
11: \hspace{3em} \textbf{else}
12: \hspace{4em} Assign a fake bridge to the user in $M_{i,j}$
13: \hspace{3em} \textbf{end if}
14: \hspace{2em} \textbf{end for}
15: \hspace{1em} \textbf{end for}
16: \textbf{end function}

\section*{Acknowledgment}

The authors would like to acknowledge supports from NSF under grants CCF-1320994, CCR-0313160, and CAREER Award 644058. We are also grateful for valuable comments from Mahnush Movahedi of the University of New Mexico.

\section*{References}


[Tor15a] The Tor Project metrics: Direct users connecting between January 1, 2015 and March 31, 2015. Available at: [https://goo.gl/mz1vLS](https://goo.gl/mz1vLS), 2015.

[Tor15b] The Tor Project: Pluggable transport. Available at: [https://goo.gl/SBGupD](https://goo.gl/SBGupD), 2015.

