On the Risks of Serving Whenever you Surf

Vulnerabilities in Tor's blocking resistance design

Jon McLachlan
University of Minnesota
Minneapolis, MN 55455
mcla0181@umn.edu

Nicholas Hopper
University of Minnesota
Minneapolis, MN 55455
hopper@cs.umn.edu

ABSTRACT

In Tor, a *bridge* is a client node that volunteers to help censored users access Tor by serving as an unlisted, first-hop relay. Since bridging is voluntary, the success of this circumvention mechanism depends critically on the willingness of clients to act as bridges. We identify three key architectural shortcomings of the bridge design: (1) bridges are easy to find; (2) a bridge always accepts connections when its operator is using Tor; and (3) traffic to and from clients connected to a bridge interferes with traffic to and from the bridge operator. These shortcomings lead to an attack that can expose the IP address of bridge operators visiting certain web sites over Tor. We also discuss mitigation mechanisms.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*

General Terms

Security

Keywords

Anonymous Communication, Blocking Resistance

1. INTRODUCTION

Tor [12] is a popular low-latency anonymity network. As of early 2009, the Tor network consists of roughly 2000 "Onion Routers" that voluntarily relay traffic between approximately 200,000 clients and the rest of the Internet. The goal of Tor is to conceal who communicates with whom: servers should not be able to determine which clients they are servicing, and "local" adversaries who can observe a Tor client's network connections and even control some Onion Routers should not be able to control or determine who the client communicates with.

While a common use of Tor is to preserve the privacy of a user accessing web services, Dingledine and Mathewson [11] have noted that a growing set of Tor users seem to be using it as a tool for censorship resistance. Since Internet censorship, the use of technical means by censoring agents to block access to sites with objectionable content, is an increasingly widespread practice [10, 7], this trend is not surprising. Because a Tor client's destination is hard to control or determine, Tor can be an effective mechanism for accessing and contributing to sites with content that some regimes may wish to censor.

However, Tor has a known weakness as a censorship resistance tool: Given that the centralized relays of Tor experience extremely little churn and are listed publicly, blocking access to Tor is as simple as downloading this list and blocking connections to the hosts it contains. While the ability to block connections exiting Tor is an important feature to prevent abuse complaints, the ability to block connections entering Tor has no corresponding justification, and means that as its popularity for providing censorship resistance grows, the likelihood that it will be blocked also increases.

To counteract this situation, the designers of Tor introduced a new method of accessing the Tor network: *bridges*. A bridge is essentially an end-user who wishes to help people who can not otherwise access the Tor servers, "bridge" themselves into the Tor network by serving as an unlisted, first-hop relay. This means that censoring adversaries now have to blacklist not only every Tor relay, but also every Tor bridge. Given the large numbers and high churn of end-users in Tor, and the goal that end-users will become bridges by default, the task of blocking Tor becomes more cumbersome.

An important question about the bridge design, raised by Dingledine and Mathewson [11], is whether running a bridge will expose clients to additional privacy risks. This is clearly a central concern, because if the cost, in terms of privacy, of running a bridge is too high, then few users will operate them, and the design will not provide the desired level of blocking resistance.

In this paper, we report on experiments that suggest that the current bridge architecture allows improved attacks on the anonymity of bridge operators. Our experiments show that two attacks previously proposed against other peer-to-peer anonymity schemes can be adapted and combined to apply successfully to Tor clients operating as bridges. We also suggest possible mechanisms to mitigate these attacks, at the expense of the level of service provided to the clients of a bridge, and show that these mechanisms effectively counteract the known attacks.

1.1 A Brief Overview

Our attacks, described in Section 3, rely on three architectural weaknesses of the bridge design:

1. **Bridges are easy to find.** Although it is a requirement that (nearly) everybody should be able to find *some* bridge, the current bridge design makes it relatively easy for an attacker to find *many* bridges. We used the Tor network to find a list

of 247 bridges; we also describe several other methods of finding a more comprehensive list of bridges.

2. **A bridge always accepts connections when its operator is using Tor.** Because of this, an attacker can compile a list of times when a given operator was either *possibly* or *certainly not* using Tor, by repeatedly attempting to connect to the bridge. This list can be used to eliminate bridge operators as candidates for the originator of a series of connections exiting Tor. We demonstrate empirically that typically, a small set of linkable connections is sufficient to eliminate all but a few bridges as likely originators.

3. **Traffic to and from clients connected to a bridge interferes with traffic to and from a bridge operator.** We demonstrate empirically that this makes it possible to test via a *circuit-clogging attack* [17, 15] which of a small number of bridge operators is connecting to a malicious server over Tor. Combined with the previous two observations, this means that any bridge operator that connects several times, via Tor, to a web-site that can link users across visits could be identified by the site's operator.

In Section 4 we discuss and evaluate mechanisms for decreasing the impact of, or eliminating, each of these architectural shortcomings: give "tokens" to access a bridge; decouple the bridge status from the operator's Tor use; and implement a scheme that isolates bridge operator traffic from bridge-relayed traffic.

1.2 Outline

The remainder of this paper proceeds as follows. We discuss the Tor anonymity scheme and related work in section 2 and the results of our experiments on Tor Bridges in section 3. We then present our mitigation mechanism and experiments analyzing its effectiveness in section 4. Finally we briefly discuss implications of these findings in section 5.

2. BACKGROUND

2.1 Tor and Bridges

Tor's main objective, like most low-latency, relay-based anonymity schemes, is to provide anonymity by concealing which client is communicating with which server [12]. Tor attempts to provide this anonymity against "local" adversaries that can observe at most one endpoint of a client-server connection, and control a small fraction of Tor nodes.

For our purposes, the Tor system consists of four node "types, " depicted in Figure 1 as well as several distinguished nodes. *Relays* (ORs) are the publicly-listed nodes that help to relay traffic between end-users and the rest of the Internet. *Clients* are nodes that connect to Tor through public relays. *Bridges* are clients that also operate as unlisted relays to allow censored nodes to access Tor; we call the "user" of a bridge the *bridge operator*. *Bridge clients* are clients who access Tor using a bridge as a first-hop relay.

Tor attempts to achieve anonymity against these local adversaries with the help of "onion relays," a collection of servers. Every OR publishes a "descriptor" listing its IP address, port, and public key, to the centralized Directory Service, which collects and manages the information of all ORs. Clients contact the DS, download a list of all ORs, and construct anonymous circuits (or tunnels) through a randomly selected set of relays.

These tunnels are built to obscure their initiator by iteratively contacting each successive relay only through the current end of

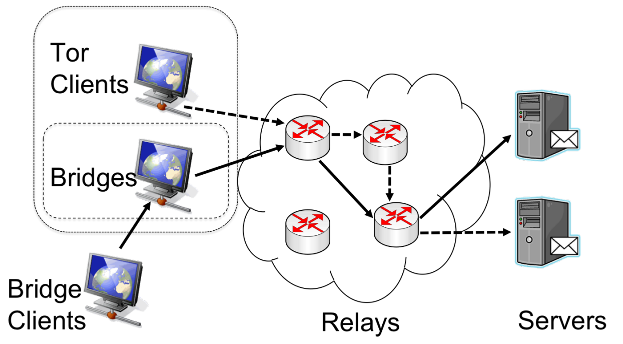


Figure 1: Tor Node types: (Public) *Relays* forward connections for others; *clients* connect to Tor using public relays; *Bridges* are clients that act as (unlisted) relays for censored nodes; *Bridge clients* access Tor via bridges.

the tunnel, and performing half-authenticated key exchange to establish a shared symmetric key between each relay in the path and the initiator. When a message traverses a tunnel from the initiator, each hop along the way removes one layer of encryption before passing the message to the next hop. At the final node, the message is in plain-text, as it has completed the entire journey through the circuit. The "exit" node of these anonymous circuits is responsible for forwarding these messages to arbitrary Internet hosts on behalf of the initiator. Responses travel back up through the sequence, with each relay encrypting the message with a new layer of encryption, back to the initiator. When the initiator receives a message from the tunnel, she removes all n layers of encryption yielding the message from the remote server.

Since the list of relays is public and centrally available, it is trivial to block access to Tor – network administrators that wish to do so can simply block traffic to the ORs' IP addresses. To counteract this, the Tor developers designed and released the bridge service in December 2007.

The bridge itself is just that: a "bridge" for otherwise blocked users which may be used to enter the Tor network. In terms of raw functionality, it is identical to an OR that does not publish its descriptor to the Directory Service. Instead, bridges publish their Bridge Descriptors to a Bridge Authority. As such, bridges can be discovered only by communication between end-users and the bridge authority. The Tor development team has provided end-users with basic means of discovering a small number of registered bridges via a website, an email, or a direct query to the Bridge Authority. Distribution of bridge information must be limited, because if the bridges are easily enumerable, then they are subject to the same blockage that the Onion Relays are subject to. However, if a given host is suspected of being a bridge, the protocol does not prevent an adversary from attempting to connect to standard ports on the host and establish a circuit.

2.2 Surfing vs. Serving

Since a bridge, in particular, supports tunnels for bridge operators and bridge clients, we distinguish two relevant states of a bridge node. A bridge is *surfing* whenever it is relaying streams over Tor that originate from its operator. A bridge is *serving* whenever it will accept connections from third party bridge clients and relay those connections over Tor. We note that functionally, bridge clients, clients, bridges, and relays all run the same Tor software, and configuration options are used to determine which role a given Tor instance will take. In particular, this means that whenever a bridge operator starts the Tor software so that she can *surf* anony-

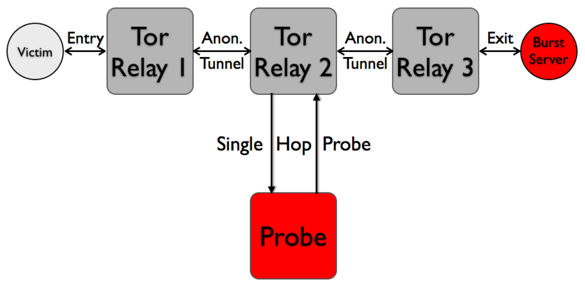


Figure 2: Circuit clogging.

mously, then her Tor instance will also be available to *serve* other clients as a bridge. Thus, architecturally, she must "serve others" in order to "surf."

2.3 Related Work

A variety of attacks have been proposed both against low-latency anonymity schemes such as Tor [1, 8, 20, 19, 17], higher-latency mixing schemes [9, 3, 14], and "peer to peer" anonymity schemes like Crowds and Salsa [22, 5, 16]. Among these, the two most closely-related attacks are *Intersection Attacks* and *Circuit-Clogging*.

2.3.1 Intersection attacks

The general concept of an intersection attack seems to have been first published by Berthold *et al.* [4]. The core idea of these attacks is that a global adversary can see which users contribute messages to a mix in each round, and which users receive messages in each round. If Alice always sends messages to Bob, then by taking the intersection of the sets of users that receive messages in rounds when Alice contributes a message, the adversary will eventually reduce the set of possible recipients to Bob. Danezis [9] extended this attack to deal with the case where Alice sends to multiple recipients, and Mathewson and Dingledine [14] observed that even in the presence of cover traffic, periods when a possible sender is offline can lead to the discovery of a sender's correspondents. These attacks assume a global adversary and were tested by simulations with synthetic output distributions and availability data. Wright *et al.* [22] extended the attack to the case of a single malicious server that could witness how long the users of an anonymity scheme were online. They showed that under certain assumptions about the behavior of users, many users of a scheme will "churn out" before a connection terminates, eliminating them as possible originators, and tested their attack with real output distributions and synthetic availability data. Note that this version of the attack critically requires knowledge of the session length.

2.3.2 Circuit Clogging

Proposed first by Murdoch and Danezis [17], circuit-clogging works as follows. Suppose that a Tor client, whom we call the "victim," connects to a malicious server over Tor. First the victim constructs an anonymous tunnel in Tor. Next the victim connects, via this tunnel, to the malicious server. To conduct the clogging attack, the server begins sending a long stream of data, modulated in an alternating pattern of long (on the order of 30 seconds) high-rate bursts of data (called "active" periods) followed by equally long "silent" periods with no traffic at all; we call the malicious server acting in this capacity the "burst server." If the data rate during active periods is high enough, then the load on the nodes of the Tor circuit will differ considerably during active and silent periods.

While the "burst server" is alternating between active and silent periods, a colluding node, the "probe" carries out the second part of the attack. The probe's job is quite simple: find all the relays whose packet-processing latency is correlated with the burst-server's on/off periods. To collect this performance data, the probe creates a one-hop anonymous circuit to itself through the targeted node. Through this one hop circuit, the probe sends simple timestamp messages. When the probe notices that the targeted node's latency consistently increases during the "active" period when compared to the "silent" periods, then this high correlation is used to probabilistically identify it as supporting the victim's anonymous circuit.

In a closely related clogging attack described by Back *et al.* [1], relay clogging, the roles of the probe and malicious server are swapped: the server times the latency of the client's anonymous tunnels, and sends high-bandwidth "clogging" streams to various relays. These streams can be sent at the network level, making the attack difficult to defend against under the current Internet architecture. However, the attack also has higher bandwidth requirements, since the high-bandwidth "clogging" packets must be sent to all suspected relays, rather than a single tunnel.

3. ATTACKING A BRIDGE

3.1 Attack Scenario

The basic scenario of our attack is as follows: suppose a Tor client, the "initiator," contributes frequently to the same website via Tor, and furthermore an attacker can link these multiple contributions, for example because the website is a pseudonymous wiki, blog, or discussion forum. We suppose that the initiator is using Tor because she wants to preserve her anonymity, and furthermore that the attacker is someone she would not want to identify her. We assume that the attacker can (1) collect a list of times that the pseudonymous operator has contributed, and (2) cause the operator to connect at least once to a server controlled by the attacker, and (3) access the Internet with moderate upload bandwidth, around 500 Kbps[1]. We note that in the case of wikis, as well as many blog and forum packages, points (1) and (2) do not requires the attacker to control the server since editing/posting times are public information and private messages or "talk" pages allow sending image URLs that are likely to be accessed by only the initiator.

Our attack proceeds in three phases, corresponding to the three architectural weaknesses identified in Section 1. In the *Bridge Discovery* phase, the attacker compiles a list of bridges using one of several methods described in Section 3.2. Since the bridge design envisions most Tor clients as bridge operators, the initiator will be in this set with reasonable probability.

In the *Bridge Winnowing* phase, many bridges are eliminated as candidate initiators. This phase exploits the observation that the nature of the bridge design – on by default – allows a remote, non-global adversary to observe whether an arbitrary bridge operator is possibly *surfing* at an arbitrary time. Several works have observed that this knowledge, along with the knowledge of some links exiting an anonymity service, could be used to conduct an intersection attack [14, 22]. We test the effectiveness of this technique with data collected from bridges and users' behaviors on Wikipedia. We show how to extend this technique so that an attacker can effectively reduce the set of possible initiators of a repeated connection to a smaller, more manageable number.

Given this smaller, set of possible initiators, we observe that the circuit-clogging attack, proposed and empirically validated by Murdoch and Danezis [17], can be applied to bridges, forming the *Bridge Confirmation* phase. In an extension of circuit clogging, McLachlan and Hopper [15] previously demonstrated that when a

[1]Assuming all 200K Tor clients become bridge operators; the bandwidth required scales linearly with the number of bridges

P2P anonymity scheme is circuit clogged, that the results are even more devastating than when applied to a centralized scheme: in P2P schemes, the victim's physical machine is also a relay, can be probed directly, and in at least one scheme has a level of correlation that is distinguishable from "middle" relays [15]. We empirically demonstrate that a slightly modified circuit clogging attack effectively confirms that a bridge operator is the true originator of a connection, without needing to probe middleman relays.

We note that while the second and third phases of our attack are similar to previously known attacks, their application to this context has several interesting aspects. In particular, the winnowing phase is similar to the long-term intersection attack, which is normally discussed in the context of a global eavesdropper, but because of the bridge setting our attack can be inexpensively mounted by a remote adversary that in many cases does not require the cooperation of either endpoint. Similarly, combining candidate identification with circuit-clogging drastically reduces the bandwidth cost of the attack, since, given a small set of candidate nodes that can be probed directly, there is no need to discover the other nodes participating in the circuit.

3.2 Discovery

It is a requirement of the bridge design that every bridge client should be able to discover some small number of bridges; otherwise the fact that bridges are hard to block does not improve the ability of censored users to circumvent blocking. On the other hand, it should not be too easy to find the list of bridges, or they become as convenient to block as the public relays. The current bridge authority design attempts to resolve this conflict by rate-limiting each 24-bit IP prefix to discover a small set of distinct bridges each week. At this rate, an attacker with few prefixes would require many discovery trials to collect a large pool of bridges.

Unfortunately, several flaws in the design of bridges and the bridge authority make it possible to quickly enumerate a larger set of bridges using a single IP address. Tor Exit nodes are not treated as sharing a single IP prefix, despite the fact that this is explicitly mentioned in the bridge design [11]. This allowed us to connect the bridge authority through each Tor Exit node and obtain a distinct set of bridges. We repeated this attack on a weekly basis from December 3 2008 through December 17 2008 and obtained 247 bridge descriptors, which form the input set for the evaluation of the winnowing attack in the next section. Note that since the set of bridges given to a single IP prefix changes weekly, and we conducted the experiment for 2 weeks, we would expect to collect only 6 bridges if the bridge authority correctly classified all Tor Exit nodes with the same IP address. A similar, one-time experiment in December 2008 using 47 open HTTP proxies found by a search engine discovered 129 of these bridge descriptors.

While we do not claim that this particular experiment represents a serious attack against bridges, an obvious question to ask about our experiment is what fraction of the available bridges it discovered. During the two-week period in which our list of bridges was discovered, 1716 distinct bridges were listed as running by the bridge authority; our experiment identified 59 of these and 140 of the 13479 additional bridges that were not available during the discovery phase but were available for contact at some earlier time.

Assuming the bridge authority makes an effort to discover and reclassify all such relay methods into a single IP prefix, a more serious issue remains. Bridges do not attempt to "hide" the fact that they are bridges. Any node that attempts to connect on a standard "bridge" port (of our 247 bridges, 197 ran on a port in the list $\{80, 443, 8080, 9001\}$) and initiate a Tor connection is likely to succeed. Thus, in the worst case a patient adversary could discover all bridges by simply polling the Internet over the course of several weeks. In addition, if the adversary has some reason to suspect that the initiator of a particular series of contributions originates in some smaller IP address range, such as the address block of a specific organization or country, the discovery process can simply poll this range instead. Another method of finding bridges that are active frequently enough to provide service to blocked users is to run a regular Tor relay that connects back to the standard "bridge ports" of any non-relay node that contacts it; a bridge will accept the connection while a regular (non-bridge) client will not. If the adversarial relay R provides fraction p of the total Tor network bandwidth (roughly 400 MBps as of June 2009) then we expect R to see a bridge after it builds or relays $1/p$ circuits. A single relay advertising the maximum 10 MBps of bandwidth[2] would thus discover 63% of the bridges that relay at least 40 circuits and 87% of the bridges that run for at least 80 circuits. Note that all Tor clients pro-actively build circuits every 10 minutes, so if a bridge has even one client it will be discovered with reasonable probability.

Finally, we note that even if an adversary does not obtain a complete list of bridges, this *only* affects the false negative rate of our attacks: if the initiator is in the set she can be identified but if she is not it does not increase the probability of falsely identifying another bridge. More precisely, because the adversary $\mathcal{A}$ cannot output node s if s is not in its input list of bridges, the bayesian probability $\Pr[I = s|\mathcal{A}(L) \Rightarrow s]$ does not depend on the probability $\Pr[s \in L]$, since

$$\begin{aligned}\Pr[I = s|\mathcal{A}(L) \Rightarrow s] &= \frac{\Pr[\mathcal{A}(L) \Rightarrow s \wedge I = s|s \in L]\Pr[s \in L]}{\Pr[\mathcal{A}(L) \Rightarrow s|s \in L]\Pr[s \in L]} \\ &= \frac{\Pr[\mathcal{A}(L) \Rightarrow s \wedge I = s|s \in L]}{\Pr[\mathcal{A}(L) \Rightarrow s|s \in L]}.\end{aligned}$$

3.3 Winnowing

The basic idea of the winnowing phase is very simple: if a Tor user has configured her software to act as a bridge, then an attacker who knows the user's IP address and bridge port can contact the bridge to determine whether the operator is "serving:" if not, she can't possibly be "surfing." Thus a patient server can check the status of a bridge at the time of each observed Tor connection. It is easy to see that if every bridge has some independent, nonzero probability of being offline, then the asymptotic probability that any client other than the user is serving at the time of every connection is 0. Furthermore, if we let ϵ be the minimum probability of going offline among a group of N bridges, and let m denote the expected number of observed Tor connections to identify a client with confidence $1 - \delta$, then we have $m < \frac{\ln N + \ln(1/\delta)}{\ln(1+\epsilon)}$.

In real life, of course, there are correlations between the online and offline behaviors of both users and bridges, caused by phenomena like "day" and "night" and "network outages." These correlations may make it impossible to differentiate between bridge operators based only on their online/offline (i.e. serving/not serving) status over a limited time frame. In order to determine the extent of correlation between bridges, we conducted a survey of the status of the 247 bridges we discovered in Section 3.2 over a 1.33Msec-period in February 2009. We checked the status of each bridge on the list every 300 seconds by attempting a connection to its bridge port. Of the 247 bridges we discovered, only 87 were active during the period of our experiment. Although we did not optimize our experiment, the cost per bridge per status check can be made

[2]Note that at the moment, Tor does not attempt to determine whether a relay can provide the level of bandwidth it advertises [2, 18], so the actual bandwidth cost can be much lower than 10MBps.

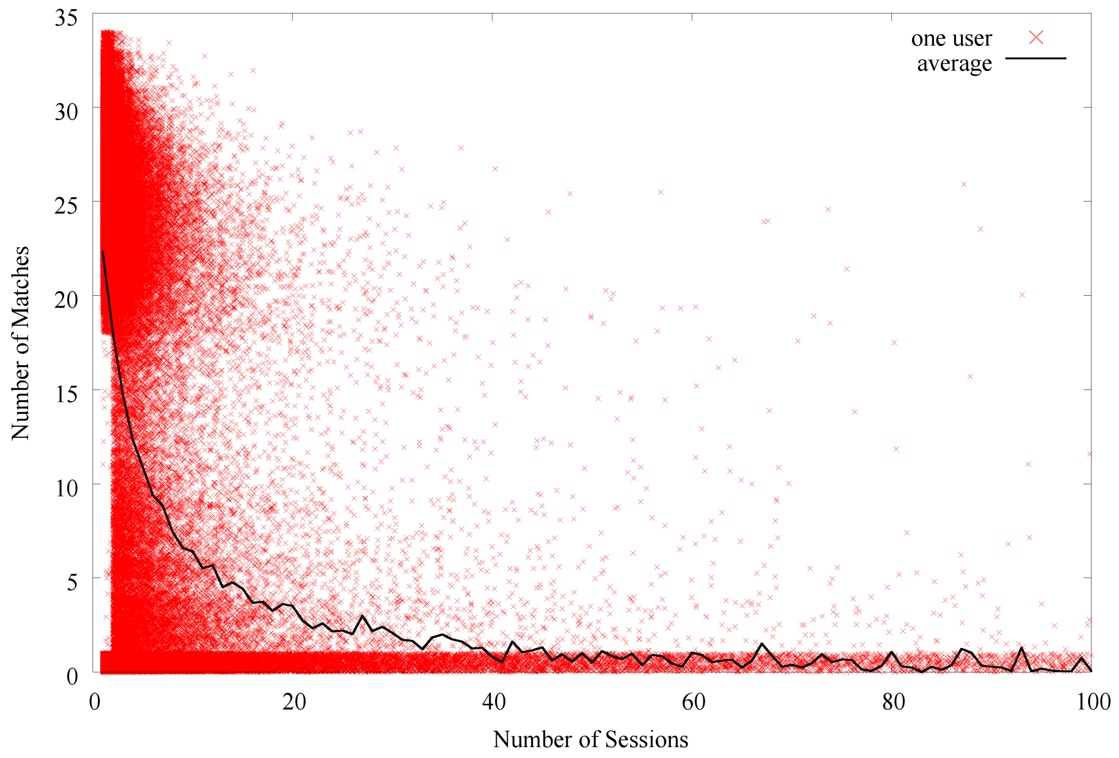


Figure 3: Results of winnowing. Each point corresponds to one user (perturbed to show density), and the black line represents the average number of bridges matching all users with a given number of sessions. Overall, roughly 23% of users do not match any bridges.

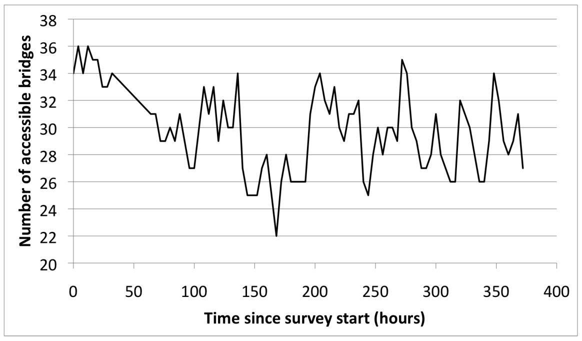


Figure 4: Results of the bridge survey. On average 29.6 of the 87 bridges surveyed were accessible.

as little 640 bits (a TCP "SYN" packet with no data, followed by a TCP "RST" packet if a "SYN/ACK" is received.) Thus the expected bandwidth cost per bridge of this phase is less than 2bps. The results of this survey are summarized in figure 4. Specifically, notice that the average number of bridges that are "serving" at any one time was only 29.6, just over 10% of the bridges returned by the authority.

To realistically model the behavior of users, we used the MediaWiki API to track the access patterns of 186,935 registered Wikipedia users over the same time period as the bridge survey. For each user, we computed the number of bridges that were "serving" at all of the times the user "touched" Wikipedia during the survey. We then grouped these "touches" into "sessions" based on their temporal proximity, in order to see the effect of repeated contributions on anonymity. We would expect that users with more contributions would have fewer matching bridges, on average. If a user has *no* matching bridges, this means that, if she had been editing Wikipedia while operating a Tor bridge, hers would be the only bridge candidate remaining after the winnowing stage.

We note that Wikipedia was used for this experiment for a reason: because Wikipedia blocks edits from Tor nodes, and has public editing data, we did not violate the privacy of any users to evaluate the effectiveness of our attack. Since the times are synchronized between the experiments the data do show the potential effect of correlations between bridges' and users' uptimes. More importantly, however, there are other online forums, such as the whistle-blower site Wikileaks, which implement the Wiki API (allowing their users to be observed) and strongly encourage postings via Tor due to the strong privacy needs of their users. Thus the results of our analysis should be of particular concern to users who contribute to such sites.

The results of our analysis appear in Figure 3. As can be seen from these results, even a few connections can significantly reduce the number of nodes that are candidate bridge operators for a pseudonym. Among all users, 23.3% matched no bridges. Among the 82274 users who were active for multiple sessions during the survey, this fraction rises to 37.3%, and among users with 50 or more sessions, 95.7% matched no bridges and only 97 of the 6204 users matched 9 or more of the 87 bridges.

Suppose the initiator uses Tor to pseudonymously blog once per day for half a year (or tweet 12 times a day for 15 days). Among our data, 2329 users had 180 or more sessions and the total number of false positives was 89; this yields a false positive rate of 0.000439. Thus even if 10000 Tor clients volunteer to bridge, on average only 4.4 bridges would be left after the winnowing stage. These data indicate that long-term use of a pseudonymous forum while operating a bridge should be strongly discouraged.

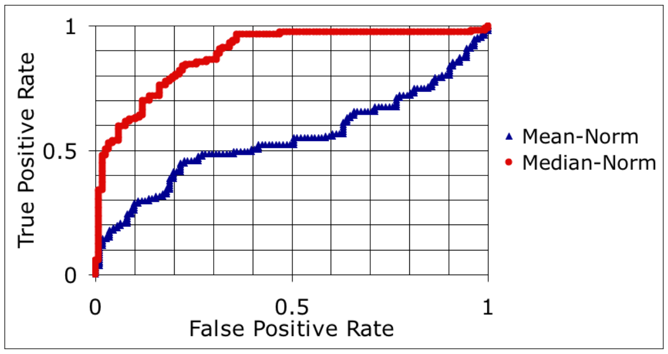


Figure 7: ROC graph demonstrating the false positive/false negative tradeoff when circuit clogging a bridge. AUC for mean-normalized latencies is 0.549874, and AUC for median-normalized latencies is 0.8840.

3.4 Confirmation

In this section, we describe our experimental environment and empirically corroborate that connections originating at a bridge can be confirmed using "circuit clogging" attack.

3.4.1 Setup

We deployed our experimental attack on Tor nodes running on a random collection of continental US PlanetLab [6] machines. All nodes were strictly disjoint from the live Tor service, except when the "victim" constructed generic 3 hop Tor circuits, the default client behavior. We deployed a controller written in Python to interface into Tor's API. This abstraction layer provided a simple means to exchange OR and bridge descriptors, construct circuits, and attach streams to those circuits. The burst server and probe were both written in C.

Figure 5 shows the basic attack scenario. In each run, a bridge operator, the "initiator" builds a circuit and connects to the burst server. The probe builds a two-hop circuit through a "victim" bridge which is either the initiator or an *innocent* bridge, that is extended to an exit server controlled by the attacker. The probe server then collects latencies while the burst server modulates the client tunnel workload. We call a run probing the initiator a "victim" run and one probing an innocent bridge a "disjoint" run.

3.4.2 Results and Analysis

Overall we performed 9 batches of 20 runs, resulting in 180 victim runs and 180 disjoint runs. For each run, we computed the correlation between the probe latency and the burst periods using the same formula as Murdoch and Danezis, that is, the correlation for bridge β was computed as

$$\chi_\beta = \frac{\sum_t B(t) L'_\beta(t)}{\sum_t B(t)} ,$$

where $B(t)$ is an indicator variable for the state of the burst server at time t and $L'_\beta(t)$ is the normalized latency of the probe through β at time t, e.g. $L'_\beta(t) = L_\beta(t)/\mu_\beta$, and μ_β was the mean latency of β,

$$\mu_\beta = \frac{1}{|T|} \sum_{t \in T} L_\beta(t) .$$

The results are shown in Figure 6(a). The distribution of MD correlations for disjoint and victim runs appear to be quite similar. Analyzing our data more closely, we found that victim runs often experienced one or more large latency spikes during "off" periods which artificially inflated the mean latency, decreasing the overall correlation. To correct for this, we recalculated the correlation using normalized latency function $L''_\beta(t)$, which was normalized by the median, rather than the mean, observed latency, as in [15]. This dramatically improved the distinguishability of the two run types, as shown in Figure 6(b).

As with any classifier, using a correlation figure to distinguish between victim and innocent bridges allows for a tradeoff in terms of false positive and false negatives: setting a higher threshold for confirmation will eliminate more false positives, but may also eliminate some true positives, and a lower threshold may expand the set of true positives that are identified while also increasing the false positive rate. Figure 7 shows the ROC curve for our results, summarizing the false positive/false negative tradeoff. The mean-normalized correlations are a weak tool for distinguishing victim and disjoint runs, with an area under curve (AUC) of 0.55, which is very close to the AUC for the nondiscriminating classifier that flips coins. However, the AUC of the median-normalized correlation classifier is 0.88 and the graph is very close to the perfect classifier, an elbow at $(0, 1)$. This indicates that circuit clogging is an effective way to confirm that a bridge is the origin of a circuit.

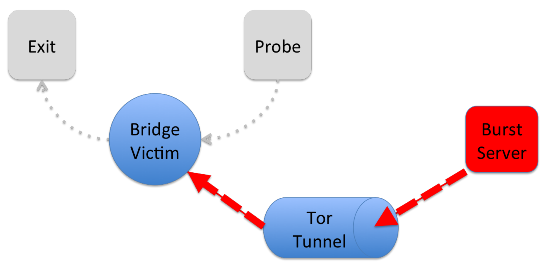
Finally, we note that repeated applications of circuit clogging can effectively amplify the false negative and positive rates in a standard way. For example, we note that the equal error rate measured in our experiments is 0.2; thus there exists a correlation threshold τ such that true positives produce correlation greater than τ with probability 0.8 while true negatives produce correlation greater than τ with probability 0.2. If the attacker repeats the clogging experiment 10 times and outputs "true" only if at least 5 runs produce correlation greater than τ, then the probability of a false positive (or, by symmetry, a false negative) is at most $\sum_{i=5}^{i=10} \binom{10}{i} (0.2)^i (0.8)^{10-i} < 0.033$. Combined with the low false positive rates of the winnowing attack, these results suggest that if a bridge is discovered in the discovery phase, it will be identified with high confidence by the combined attack, and if it is not discovered, with high probability the attack will detect this, and output zero matching bridges.

4. MITIGATION TECHNIQUES

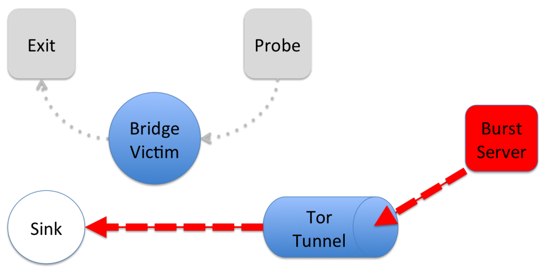
In this section, we discuss strategies for mitigating all three attack phases, without altering the architecture of the bridge service. The discovery phase is difficult to address fully without more radical changes to the bridge architecture, but we discuss some "band-aid" mechanisms that can be applied. The winnowing phase relies on a bridge operator's serving whenever she surfs – the default behavior for bridges – so we consider several strategies to weaken this assumption. In the case of circuit clogging, it is well understood that the attack works because of information leakage that occurs at the relays in the form of interference between disjoint anonymous circuits (the "modulated stream" and "probe stream") that pass through the same relay. We describe a mechanism that greatly reduces this interference in a bridge.

Before discussing these mechanisms in detail, however, we point out that one "fool-proof" but unsatisfying solution exists: *never serve where you surf.* Completely separating "server" activities from "client" activities eliminates the threats to the bridge operator's privacy, and extends to any other services a user might want to offer on the public Internet, which might also serve as targets of a winnowing attack. However, this also defeats the purpose of the bridge design, which is to create a large, dynamic pool of nodes that must be blocked: if the set of bridges is limited to a static set of nodes, then bridges become nearly as easy to block as relays.

One seemingly obvious defense mechanism for circuit-clogging is to enforce a strict "exclusive or" rule as follows: Either the bridge

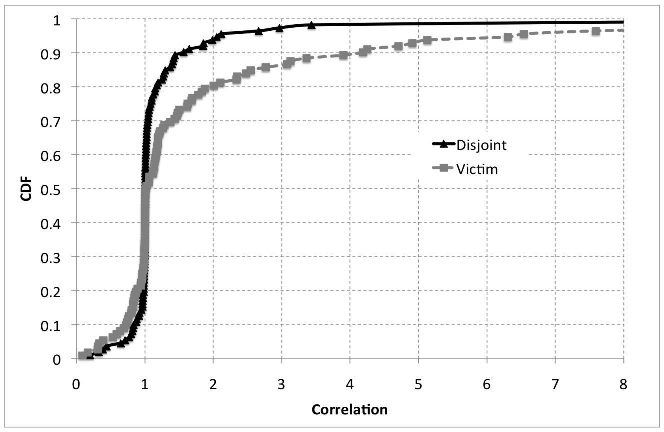


(a)

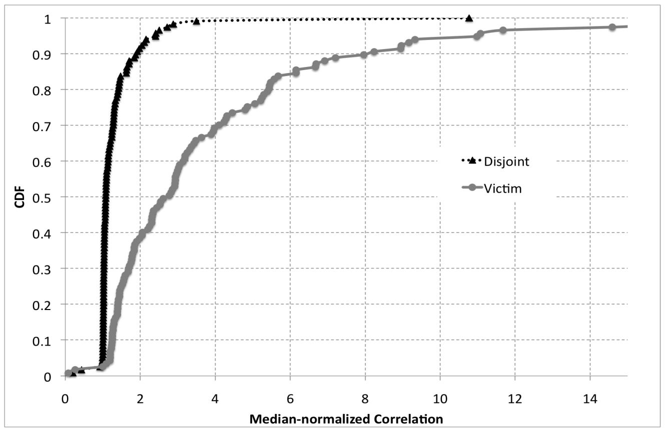


(b)

Figure 5: Bridge clogging experimental setup showing (a) intersecting and (b) disjoint configurations. Arrows indicate direction of attack traffic flow.



(a)



(b)

Figure 6: CDF of correlations from 180 runs: (a) mean-normalized, and (b) median-normalized.

can support a bridge client or a bridge operator, but not both. Of course this always results in one unserviced user when both want to use the system - not to mention a race condition to the construction of the first tunnel. Although this prevents the bridge from being used by the probe and malicious server simultaneously, the usability implications are not acceptable, even if we were to give the bridge operator priority.

Our mitigation mechanism still allows for bridge operators to help censored users overcome various communicative repressions. We minimize the amount of application level interference between tunnels created by the bridge operator and tunnels created by bridge clients. We (1) group together all circuits from the bridge operator and all circuits from bridge client(s) and then (2) enforce strict application level processing policies between the two groups of circuits to minimize the latency information leakage between them.

Notice that this bridge-specific mitigation mechanism allows the bridge operator to specify the processing policies which will be enforced. We implemented one specific policy which gives the operator the ability to set the percentage of the CPU that will service local anonymous circuits and the percentage of the CPU that will service bridge circuits. It is interesting to note that while the defense mechanism proposed by [15] merely *increases the cost* of attack by stochastically separating the attacker's streams, our mechanism essentially eliminates the attack, and its variants, altogether.

4.1 Mitigating bridge discovery

One obvious step that the bridge authority can take to try to improve rate-limiting is to attempt to classify all relaying nodes, including Tor relays, other anonymity or caching services, and any open proxies, as sharing a single IP prefix. Unfortunately, there does not seem to be a foolproof method of discovering all such services. One approach that seems promising would be a heuristic that attempts a proxy connection to any IP address that makes a discovery query. If the connection succeeds, then the requester is using a proxy. We note that other heuristics with some false positive rate should be permissible as well - if a node is misclassified as a proxy it simply will get the same set of bridge descriptors as other proxy nodes, which should still serve the purpose of bridging censored users to the network.

The problem of a "polling" adversary could be partially mitigated by requiring a client that connects to the bridge port to "prove" that it knows the bridge, by e.g. first sending a HTTP "GET" request with the hash of the bridge's public key over the client-bridge SSL connection.[3] This would prevent attackers that have not discovered the bridge's descriptor through the Bridge Authority from receiving absolute confirmation that a given IP address is a bridge. We note that as described by Dingledine and Mathewson [11], further measures may be necessary to resist statistical classification of bridges based on their reaction to unauthenticated connections.

One method which would significantly raise the cost of scanning or polling attacks would be to have bridges listen on random ports rather than the standardized 443. The bridge design supports this option, but running on the standard `https:` port is an import feature for blocking resistance: assuming the censor allows *any* web connections at all, these ports are the least likely to be blocked at the network level. Thus moving to other port numbers would not appear to be a viable solution for the needs of most censored users.

[3]Clearly, adopting this suggestion means that a bridge MUST NOT support Tor's "certificates up front" TLS handshake mode, in which the relay gives a two-certificate chain where the first certificate is the relay's self-signed public key. Fortunately, recent versions of Tor support TLS handshaking with a TLS-only (single-key) cert.

4.2 Mitigating bridge winnowing

The “success” of the winnowing requires three factors:

- A user’s repeated visits to the malicious service are linkable. Techniques to reduce linking across sessions, such as disabling cookies, client-side scripting, refresh tags, and caching; using privoxy or TorVM; and others can be applied, and are recommended as standard practice when using an anonymity service such as Tor, to limit the cases in which this is possible. However, for some services such as pseudonymous use of online forums, this factor may be unavoidable.

- Different bridges will be serving when the user accesses the malicious service at different times. We can of course enforce a large “minimal anonymity set” of bridges which a bridge operator requires to be serving before she anonymously surfs. However, this seems to be an unacceptable consequence to the operator, since for even moderately large sets we would expect the probability of all bridges being simultaneously online to be quite small.

- The malicious server can connect to the bridge service whenever the bridge operator is surfing via Tor. This is the simplest assumption to invalidate from the operator’s point of view: the bridge can sometimes choose not to *serve* other clients while the operator is *surfing*. Implementing such a mechanism would trade availability to bridge clients for privacy of bridge operators.

We did not implement any of these countermeasures, because each seems to involve an unacceptable loss of usability to either bridge operators or bridge clients, compared with somewhat unclear benefits in terms of privacy. However, we can sketch a solution based on the third point above: The goal is to ensure that the probability a user’s bridge is serving is the same in case she is surfing and in case she is not. Thus, when a bridge is first enabled, it would collect several weeks’ worth of data to measure the fraction of time, f that the operator’s Tor instance is running but the operator has no active connections over Tor. During this “trial period” the bridge would never accept client connections, regardless of whether the operator is surfing.

After the trial period, the bridge would function as usual, except that when the operator begins surfing, the bridge flips a biased coin: with probability f it continues to serve, and with probability $1 - f$ it shuts down until some time after the user has stopped surfing. In this way, the operator’s “serving” status can be made independent of her “surfing” status to the malicious web server. We note that the critical detail here is that *the probability of “going offline” is identical whether the operator is surfing or not*. Thus the mere fact that a bridge becomes unavailable does not signal to the adversary that the operator is surfing. Formally, the trial period measures the probability $f = \Pr[\mathsf{serve}|\neg\mathsf{surf}]$, and the biased coin flip ensures that $\Pr[\mathsf{serve}|\mathsf{surf}] = f$. Thus $\Pr[\mathsf{serve}|\mathsf{surf}] = \Pr[\mathsf{serve}|\neg\mathsf{surf}]$ and since $\Pr[\mathsf{surf}] + \Pr[\neg\mathsf{surf}] = 1$, we have $\Pr[\mathsf{serve}|\mathsf{surf}] = \Pr[\mathsf{serve}]$, so that serving and surfing are statistically independent signals. However, care must be taken so that second-order effects are not introduced: for example, “not-serving” times caused by the operator’s surfing activities should be reasonably consistent with the length of the bridge’s downtimes during the trial period as well.

We note that several other possible methods of apparently decoupling surfing from serving are not effective. The first is to run two separate Tor instances on the same machine, one to serve as a bridge and the other as a client, and have the bridge run at all times. The reason that this method does not work is that the client will

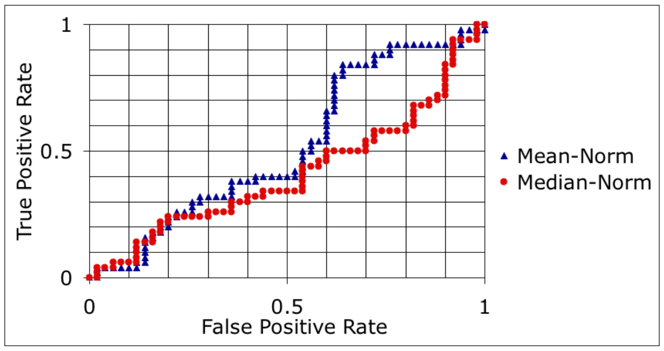


Figure 9: ROC graph demonstrating the predictive power of latency correlations for an “unfairly queued” bridge. AUC is 0.5216 **for mean-normalized latencies, and** 0.4120 **for median-normalized latencies.**

go offline ocassionally, due to hardware failures, network outages, and so on, leading to periods where, because the bridge instance is unavailable, the adversary can conclude that the client is not surfing. Mathewson and Dingledine [14] show that this coarse level of information is sufficient for statistical disclosure attacks even if the fraction of time spent offline is relatively small. The second is to configure the bridge to serve only with some fixed probability, for example 0.25. In this case the winnowing attack can be modified to become a search for a bridge that is serving during precisely 25% of the observed sessions; assuming independence only the initiator will satisfy this search given a sufficient number of observed sessions. (We note however that this solution does increase the number of observations necessary to obtain a given false positive rate.)

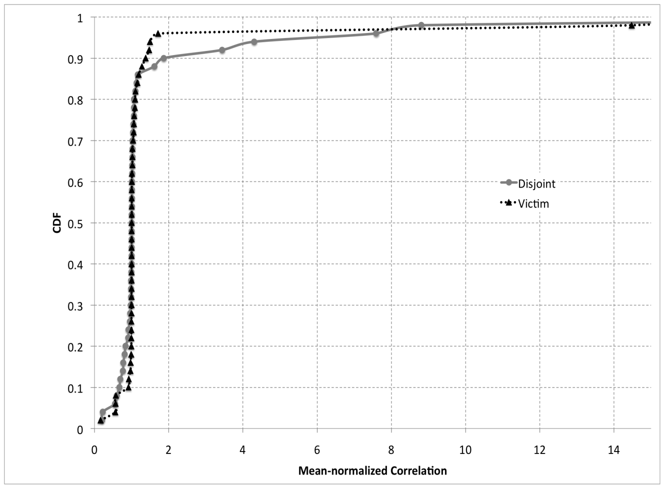
4.3 Mitigating bridge clogging

As emphasized by previous work on the circuit clogging attack [17, 15], the success of circuit clogging depends on the fact that disjoint circuits interfere with the service provided to each other - when one circuit is busy, other circuits exhibit higher latency. In the case of the MorphMix P2P anonymity scheme, McLachlan and Hopper proposed to deal with this by use of a “fair queuing” mechanism that stochastically enforced noninterference of different streams going through a relay. However, in the bridge scenario there is no reason to treat bridge operator and bridge client traffic fairly. Thus, we propose to mitigate bridge clogging with an “unfair” queuing mechanism that treats these two sources of traffic separately.

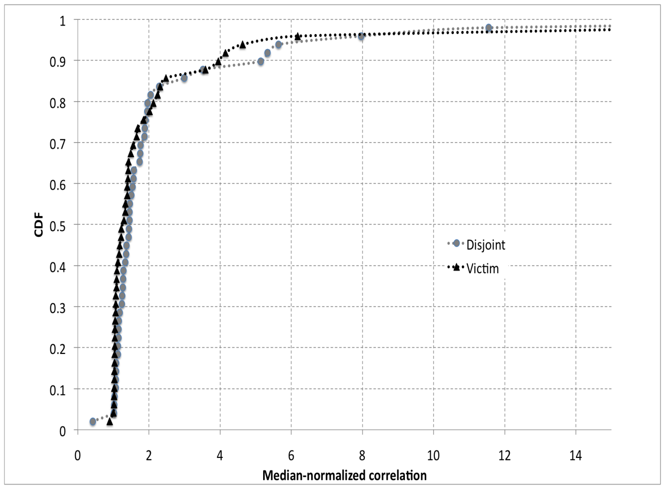
Specifically, we modified a bridge to include an “unfairness” parameter $\tau \in (0, 1)$. The bridge then divides incoming cells into two queues based on their stream ID, one that contains cells from streams that terminate locally, and one for all other streams. The bridge then allocates its processing time to these queues based strictly on the system clock: for τ fraction of the time, only cells in the local (operator) queue can be processed, and for $1 - \tau$ fraction of the time, only cells in the other queue are serviced. If at some point the currently active queue is empty, the system busy waits until either it is nonempty or the active queue changes. This enforcement based on time allocation ensures that there is very little interference between streams carried over the bridge and streams originated by the bridge operator, and allows operators to adjust the level of service they are willing to donate to help “blocked” users access Tor.

4.3.1 Evaluation.

To test the effectiveness of our bridge clogging mitigation, we conducted a batch of 20 distributed experiments on PlanetLab. Each



(a)



(b)

Figure 8: CDF of correlations for mitigated bridge clogging. (a) Mean-normalized, (b) Median-normalized.

experiment involved a similar setup to the experiments described in section 3.4: A "victim" bridge running our modified code with unfairness $\tau = 0.9$ was started on a randomly chosen PlanetLab node, and a "disjoint" bridge with the same settings was started on another randomly chosen PlanetLab node. The "probe client" made single hop connections through each bridge to the "probe server," and collected latency data while the victim bridge built a Tor circuit and connected to the "burst server." After each run, the adjusted correlation from section 3.4 was computed for both probe circuits.

The results of these experiments are summarized in Figure 9. The ROC curve based on adjusted correlation gives an area under curve of 0.41, and based on median correlation gives an AUC of 0.52, indicating that circuit clogging has only a slight advantage over random guessing in distinguishing victim bridges from disjoint bridges running the same code. Further details in the form of CDFs of median and mean-normalized correlations are shown in Figure 8, which illustrate the difficulty of distinguishing between victim and disjoint runs using "unfair queuing".

4.3.2 Efficiency and limitations of mitigation.

One obvious drawback of this mitigation mechanism is that it can lead to wasted resources. This can happen both when the bridge operator has no active circuits and bridge clients are active, and when the bridge has no active client circuits and the bridge operator has many active circuits. Since bridge operators can choose to limit the outgoing bandwidth to clients, it does not seem to be inconsistent to allow them to also limit the computational resources expended for clients; perhaps the bigger concern is the converse situation. However, the maximum loss in operator service is both easy to compute (at most $1 - \tau$ performance loss) and is adjustable based on the operator's preferences.

One natural question arising from our use of queuing-style mechanisms is whether other "quality of service" (QoS) resource allocation schemes can similarly mitigate the attack while resulting in less wasted resources. Such a mechanism would clearly be desirable, but we believe it would be unlikely to provide the desired level of security: any mechanism that both (a) gives priority to bridge operator traffic and (b) attempts to maximize resource usage seems likely to fall prey to some form of the clogging attack. This is because the "bursty" traffic from the server is destined to the bridge operator, and thus will have high priority. So the service available to "bridge clients" will differ between "on" and "off" periods, and a mechanism that detects this availability will function similarly to the "probe server" used by straightforward circuit clogging.

Another natural question that might arise is whether processing time is the correct resource to allocate between the two types of traffic; since they must share bandwidth as well this is another potential resource to control. Unfortunately, the current architecture of Tor prevents us from correctly using bandwidth controls in this way: there is no method to throttle incoming flows, so we cannot control the allocation of incoming bandwidth. And once a cell has been processed, the two types of traffic no longer contend for the same resource: operator traffic is destined for the local bus whereas client traffic must leave on a network link.

We stress that this last point is crucial: *traffic flowing to the bridge operator does not contend for "upload" bandwidth.* Thus any traffic that an attacker "pulls" from a bridge cannot interfere with the traffic the bridge operator downloads. In addition, the bandwidth required for latency probes is very small, and bridges are intended to be lightly loaded. Thus bandwidth-only controls on bridge-to-client flows are unlikely to be an effective countermeasure to circuit clogging.

We do note however that our mechanism only divides service to cells after they have been transmitted to the bridge and decrypted over TLS. This leaves the possibility that other attacks could exist based on interference between the bandwidth of incoming flows; thus some upstream bandwidth-regulation method may be required to ensure complete noninterference. The use of end-to-end congestion mechanisms, as proposed by Reardon and Goldberg [21] may allow such controls, but use of congestion control for this purpose may expose other information leaks, such as allowing the bridge's entry node to distinguish between bridge-originated streams and bridge-relayed streams. Determining the extent of this tradeoff is an important question for future work.

5. CONCLUSION

In this paper, we have initiated the study of the impact of Tor's bridges on the privacy of bridge operators. We found that several aspects of the design enable troubling attacks, which can be partially mitigated by mechanisms described here. While bridges have the potential to play a critical role in assuring freedom of speech and freedom of information, this potential must be carefully weighed against the negative implications they have for the privacy of their operators.

It is interesting to note that all of the candidate mitigation mechanisms we have discussed fundamentally require a reduction in the level of service provided to the clients of bridges in order to improve the privacy of their operators. This is unfortunate, but the mechanisms discussed also provide some minimum level of service as well, and it seems that in this setting, "some uncensored bits" are much better than "no uncensored bits."

Acknowledgments

We thank our shepherd, Steven Murdoch, for helpful suggestions regarding the presentation of our attack; Chris Arnold and Tim Stucki for sharing the results of their open proxy survey; Roger Dingledine and Nick Mathewson for clarifying some aspects of the bridge design; and Karsten Loesing for providing detailed statistics about the coverage of our bridge discovery attack. This research was supported by the NSF under grant CNS-0546162.

6. REFERENCES

[1] BACK, A., MÖLLER, U., AND STIGLIC, A. Traffic analysis attacks and trade-offs in anonymity providing systems. In *Proceedings of Information Hiding Workshop (IH 2001)* (April 2001), I. S. Moskowitz, Ed., Springer-Verlag, LNCS 2137, pp. 245–257.

[2] BAUER, K., MCCOY, D., GRUNWALD, D., KOHNO, T., AND SICKER, D. Low-resource routing attacks against Tor. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2007)* (Washington, DC, USA, October 2007).

[3] BERTHOLD, O., AND LANGOS, H. Dummy traffic against long term intersection attacks. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2002)* (April 2002), R. Dingledine and P. Syverson, Eds., Springer-Verlag, LNCS 2482.

[4] BERTHOLD, O., PFITZMANN, A., AND STANDTKE, R. The disadvantages of free MIX routes and how to overcome them. In *Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability* (July 2000), H. Federrath, Ed., Springer-Verlag, LNCS 2009, pp. 30–45.

[5] BORISOV, N., DANEZIS, G., MITTAL, P., AND TABRIZ, P. Denial of service or denial of security? How attacks on reliability can compromise anonymity. In *Proceedings of CCS 2007* (October 2007).

[6] CHUN, B., CULLER, D., ROSCOE, T., BAVIER, A., PETERSON, L., WAWRZONIAK, M., AND BOWMAN, M. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev. 33*, 3 (2003), 3–12.

[7] CRANDALL, J. R., ZINN, D., BYRD, M., BARR, E., AND EAST, R. Conceptdoppler: a weather tracker for internet censorship. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security* (New York, NY, USA, 2007), ACM, pp. 352–365.

[8] DAI, W. Two attacks against freedom. online essay: `http://www.weidai.com/freedom-attacks.txt`.

[9] DANEZIS, G. Statistical disclosure attacks: Traffic confirmation in open environments. In *Proceedings of Security and Privacy in the Age of Uncertainty, (SEC2003)* (Athens, May 2003), Gritzalis, Vimercati, Samarati, and Katsikas, Eds., IFIP TC11, Kluwer, pp. 421–426.

[10] DEIBERT, R. J., PALFREY, J. G., ROHOZINSKI, R., AND ZITTRAIN, J., Eds. *Access Denied: The Practice and Policy of Global Internet Filtering*. MIT Press, 2008.

[11] DINGLEDINE, R., AND MATHEWSON, N. Design of a blocking-resistant anonymity system. `https://www.torproject.org/svn/trunk/doc/design-paper/blocking.html`, May 2008. Online Draft.

[12] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. F. Tor: The second-generation onion router. In *13th USENIX Security Symposium* (August 2004).

[13] LOESING, K. Measuring the Tor Network: Evaluation of Bridges and Their Usage in Tor. published online: `http://git.torproject.org/checkout/metrics/master/report/bridges/bridges-2009-04-04.pdf`, 2009.

[14] MATHEWSON, N., AND DINGLEDINE, R. Practical traffic analysis: Extending and resisting statistical disclosure. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)* (May 2004), vol. 3424 of *LNCS*.

[15] MCLACHLAN, J., AND HOPPER, N. Don't clog the queue: Circuit clogging and mitigation in P2P anonymity schemes. In *Proceedings of Financial Cryptography (FC '08)* (January 2008).

[16] MITTAL, P., AND BORISOV, N. Information leaks in structured peer-to-peer anonymous communication systems. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS 2008)* (Alexandria, Virginia, USA, October 2008), P. Syverson, S. Jha, and X. Zhang, Eds., ACM Press, pp. 267–278.

[17] MURDOCH, S. J., AND DANEZIS, G. Low-cost traffic analysis of tor. *IEEE SP 00* (2005), 183–195.

[18] MURDOCH, S. J., AND WATSON, R. N. M. Metrics for security and performance in low-latency anonymity networks. In *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)* (Leuven, Belgium, July 2008), N. Borisov and I. Goldberg, Eds., Springer, pp. 115–132.

[19] MURDOCH, S. J., AND ZIELIŃSKI, P. Sampled traffic analysis by internet-exchange-level adversaries. In *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)* (Ottawa, Canada, June 2007), N. Borosov and P. Golle, Eds., Springer.

[20] RAYMOND, J.-F. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability* (July 2000), H. Federrath, Ed., Springer-Verlag, LNCS 2009, pp. 10–29.

[21] REARDON, J., AND GOLDBERG, I. Improving Tor using a TCP-over-DTLS Tunnel. In *18th annual USENIX Security Symposium* (2009). to appear.

[22] WRIGHT, M., ADLER, M., LEVINE, B. N., AND SHIELDS, C. Defending anonymous communication against passive logging attacks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy* (May 2003).