Point Break: A Study of Bandwidth Denial-of-Service Attacks against Tor

Rob Jansen, *U.S. Naval Research Laboratory;* Tavish Vaidya and Micah Sherr, *Georgetown University*

https://www.usenix.org/conference/usenixsecurity19/presentation/jansen

This paper is included in the Proceedings of the 28th USENIX Security Symposium.

August 14–16, 2019 • Santa Clara, CA, USA

978-1-939133-06-9

Open access to the Proceedings of the 28th USENIX Security Symposium is sponsored by USENIX.

Point Break: A Study of Bandwidth Denial-of-Service Attacks against Tor

Rob Jansen
U.S. Naval Research Laboratory
rob.g.jansen@nrl.navy.mil

Tavish Vaidya
Georgetown University
tavish@cs.georgetown.edu

Micah Sherr
Georgetown University
msherr@cs.georgetown.edu

Abstract

As the Tor network has grown in popularity and importance as a tool for privacy-preserving online communication, it has increasingly become a target for disruption, censorship, and attack. A large body of existing work examines Tor's susceptibility to attacks that attempt to block Tor users' access to information (e.g., via traffic filtering), identify Tor users' communication content (e.g., via traffic fingerprinting), and de-anonymize Tor users (e.g., via traffic correlation). This paper focuses on the relatively understudied threat of denial-of-service (DoS) attacks against Tor, and specifically, DoS attacks that intelligently utilize bandwidth as a means to significantly degrade Tor network performance and reliability.

We demonstrate the feasibility of several bandwidth DoS attacks through live-network experimentation and high-fidelity simulation while quantifying the cost of each attack and its effect on Tor performance. First, we explore an attack against Tor's most commonly used default bridges (for censorship circumvention) and estimate that flooding those that are operational would cost $17K/mo. and could reduce client throughput by 44% while more than doubling bridge maintenance costs. Second, we explore attacks against the TorFlow bandwidth measurement system and estimate that a constant attack against all TorFlow scanners would cost $2.8K/mo. and reduce the median client download rate by 80%. Third, we explore how an adversary could use Tor to congest itself and estimate that such a congestion attack against all Tor relays would cost $1.6K/mo. and increase the median client download time by 47%. Finally, we analyze the effects of Sybil DoS and deanonymization attacks that have costs comparable to those of our attacks.

1 Introduction

Tor [28] is the most popular anonymous communication system ever deployed, with an estimated eight million daily active users [59]. These users depend on Tor to anonymize their connections to Internet services and distributed peers, and also to circumvent censorship by local authorities that control network infrastructure. Tor is used by ordinary citizens and businesses to protect their privacy online, by journalists and activists to more freely access and contribute digital content [7], and by criminals to perform illegal activities while avoiding identification [67].

As a result of its popularity, open-source codebase [9], and transparent development processes [10], Tor has gained significant attention from researchers who explore attacks that aim to deanonymize its users by gaining an advantageous view of network traffic [13]. Research directions for Tor attacks include website fingerprinting [18, 41, 42, 56, 69, 70, 77, 84, 85], routing [15, 16, 79, 81, 83], end-to-end correlation [54, 57, 58, 65, 66], congestion [31, 34, 51, 64], and side channels [43, 63]. Many of these attacks represent realistic threats for Tor users: some attacks are reported to have been launched by state sponsors against Tor in the wild [21, 22, 73, 76]. However, relatively understudied but arguably more viable is the threat of denial-of-service (DoS).

The Threat of Denial-of-Service: Bandwidth-based DoS against Tor is a relatively understudied but relevant threat. Previous work has explored the exhaustion of Tor relays' memory [51], CPU [14, 71], and socket descriptor resources [35], as well as selective service refusal [16]. While bandwidth-based DoS attacks against a *single target* have been considered [31], we are the first to study the feasibility, cost, and effects of launching such attacks against the *entire Tor network* of relays and other particularly vulnerable Tor components. Given Tor's limited resources and slow performance relative to the open web, further reducing performance through bandwidth DoS attacks also has the potential to reduce security by driving away users who may be unwilling to endure even slower load times [16, 25].

We argue that DoS attacks in general, and bandwidth DoS attacks in particular, are less complex and therefore more viable than many previous deanonymization attacks. Our DoS attacks either can be outsourced to third party "stresser" services that will flood a target with packets for an amortized cost of $0.74/hr. per Gbit/s of attack traffic (see §3.1), or utilize lightweight Tor clients running on dedicated servers at an amortized cost of $0.70/hr. per Gbit/s of attack traf-

fic (see §3.2). Nation-states are known to sponsor DoS attacks [60], and the ease of deployment and low cost of our attacks suggest that state actors could reasonably run them to disrupt Tor over both short and long timescales. We speculate that nation-states may, e.g., choose DoS as an alternative to traffic filtering as Tor continues to improve its ability to circumvent blocking and censorship [32]. Non-state actors could also reasonably deploy the attacks since they require only a few servers or can be completely outsourced.

Tor DoS attacks are not a hypothetical threat: existing evidence indicates that DoS attacks against the network have already been successfully deployed [23, 38, 40], requiring Tor to develop a subsystem to mitigate its effects [24, 39] (the subsystem does not mitigate our attacks). Although it may be challenging to detect and counter bandwidth-based DoS attacks, especially those that are designed to mimic realistic and plausible usage patterns, we believe that it is imperative to better understand such a threat as we develop defenses.

Our Contributions: This paper focuses on the costs and effects of DoS attacks that intelligently utilize bandwidth as a means to significantly degrade Tor performance and reliability. Following a discussion of the current pricing models for "stresser" (i.e., DoS-for-hire) services (§3.1) and dedicated servers (§3.2), we first explore the threat of a naïve flooding attack in which an adversary uses multiple "stresser" accounts to consume Tor relays' bandwidth by flooding them with packets (§4). We estimate that the cost to carry out such an attack against the entire Tor network is $7.2M/mo.

We then demonstrate the feasibility and effects of 3 major bandwidth DoS attacks against Tor in order of decreasing cost. First, we explore in §5 an attack that attempts to disrupt Tor's censorship circumvention system by flooding Tor's default bridges with packets, thereby causing bridge users to migrate to non-default bridges or lose access to Tor. We estimate that flooding Tor's 12 operational default bridges would cost $17K/mo. and would reduce bridge user throughput by 44% (if 25% of the users migrated to other bridges) while more than doubling meek bridge maintenance costs.

Second, we explore in §6 an attack that attempts to disrupt Tor's load balancing system by flooding TorFlow bandwidth scanners with packets, thereby causing inaccurate and inconsistent relay capacity measurement results. Through high-fidelity network simulation using Shadow [47], we find that such an attack reduces the median client download rate by 80%. We estimate that a constant flooding attack against Tor's 5 TorFlow scanners would cost $2.8K/mo.

Third, we explore in §7 an attack that uses the Tor protocol to consume relay bandwidth resources. In the attack, a Tor client builds thousands of 8-hop circuits and congests relays by downloading large files through the network. Using Shadow, we find that such an attack using 20k circuits increases the median client download time by 120% at an estimated cost of $6.3K/mo. and achieves a bandwidth amplification factor of 6.7. We also find that a stop reading strategy [51] reduces the estimated cost of a 20k circuit attack to $1.6K/mo., increases the median client download time by 47%, and achieves a bandwidth amplification factor of 26.

Finally, we analyze in §8 the effects of relay Sybil attacks that have costs comparable to those of our attacks.

Ethics and Responsible Disclosure: We emphasize that we do not carry out attacks against the live Tor network. We conduct some measurement experiments on Tor to better understand its composition and performance characteristics. However, we neither observe nor store any information about any Tor users (other than ourselves). We evaluate our attacks using high-fidelity Shadow simulations that are constructed to resemble the live Tor network. Additionally, we discussed our project with Tor developers, shared some of our results before submission of this paper, and sent a pre-print of our paper prior to its acceptance. We anticipate providing support as they develop any mitigations to our attacks.

2 Related Work

In this paper we focus specifically on attacks that target the Tor network, noting that attacks that target Internet protocols (e.g., TCP) or resources (e.g., web servers) have been rigorously studied in previous work.

Anonymity Attacks against Tor: There is a large body of work that examines attacks against Tor. The majority of these attacks aim to compromise anonymity—that is, to de-anonymize either targeted users or Tor users *en masse*. We highlight that research directions for anonymity attacks include website fingerprinting [18, 41, 42, 56, 69, 70, 77, 84, 85], routing [15, 16, 79, 81, 83], end-to-end correlation [54, 57, 58, 65, 66], congestion [31, 34, 51, 64], and side channels [43, 63]. (The reader may refer to previous work for a more complete taxonomy [13].) Although anonymity attacks are certainly problematic for Tor, the primary focus of this paper is on bandwidth-based DoS attacks that significantly degrade Tor network performance and reliability. Note that we compare our attacks to Sybil attacks that can be used for deanonymizing Tor users in §8.

Denial-of-Service Attacks against Tor: We are not the first to explore the network's susceptibility to DoS. In their seminal work, Evans et al. [31] exploit the lack of an upper bound on the length of Tor circuits in older Tor versions. They show that an attacker can perform a bandwidth amplification DoS attack by creating cyclic, arbitrary length circuits through high bandwidth Tor relays. The congestion created by the DoS attack affects the latency of legitimate circuits which can be used to determine the guard relay on a circuit. To mitigate this attack, Tor has since imposed a cap of eight relays in circuit creation [27, §5.6].

Similarly, Pappas et al. [71] propose an asymmetric, amplification *packet-spinning* DoS attack against legitimate Tor relays. The goal of the attack is to increase the chances of legitimate clients choosing the attacker's relays by keeping the legitimate relays busy with expensive cryptographic opera-

tions. The attacker uses a malicious relay to create a circular Tor circuit that starts and ends at the malicious relay. Their focus is on de-anonymization and they do not consider DoS attacks against the entire Tor network.

Borisov et al. [16] show that an attacker can de-anonymize a large fraction of Tor circuits by performing selective DoS on honest Tor relays to increase the probability that the attacker's relays will be chosen as guard and exit relays (and thus capable of performing traffic correlation [80]). Tor has since deployed a route manipulation (path bias) detection system to mitigate the effects of such an attack [26, §7].

Barbera et al. [14] propose an asymmetric DoS attack against Tor relays. The attack floods a targeted relay with CREATE cells that require public key operations to decrypt the cell. They show that by strategically targeting important relays, the attacker can slow down the entire Tor network due to overload on the remaining relays that are not under DoS attack. This is similar in aim to our work. However, our focus is less on protocol vulnerabilities and more on enumerating hotspots in Tor that, when attacked, could disrupt the network at large. We explore multiple avenues for causing network-wide performance and disruption.

Geddes et al. [35] demonstrate socket exhaustion attacks against various proposed replacements [12, 37] of Tor's transport protocol. They show that an attacker can disable arbitrary relays by exhausting their socket file descriptors and prevent legitimate connections from succeeding. However, the attack does not apply to the deployed Tor network, which does not employ the vulnerable transport protocols.

Jansen et al. [51] propose the *Sniper Attack*, a memory-based DoS attack that exploits Tor's end-to-end reliable data transport to consume memory by filling up the application level packet queues. Using simulation on Shadow [47], they show that an attacker can sequentially disable 20 exit relays in 29 minutes and make the Tor network unusable, while remaining undetected. The Tor Project has since rolled out defenses against the sniper attack [45]. We use some of the techniques from the sniper attack in our congestion attacks.

3 Threat Model and Attacker Costs

We consider an attacker who is determined to deny service to the Tor network. We make few assumptions about the capabilities or makeup of our adversary. In particular, our adversary need not control large regions of the Internet or be able to observe a large fraction of Tor traffic.

Instead, we model an adversary who has some bandwidth and computing capacity at its disposal. Certainly, a nation-state has such resources, but we imagine that such an adversary would likely prefer to avoid attribution and not conduct attacks from its own networks. More generally, our adversary can acquire (or rent) a distributed network of machines capable of sending traffic into the Tor network. We highlight two potential avenues for obtaining the resources necessary to carry out network-wide attacks against Tor: dedicated DoS "stresser" services (§3.1) and the use of more traditional (and legal) dedicated hosting services (§3.2).

We do not require that the adversary be able to position itself in arbitrary locations on the Internet, although we do assume that some portion of its traffic will reach its intended targets. For some attacks, we additionally require the adversary to operate a Tor relay, but as we describe below, it is advantageous for such attacks to run a relay that provides negligible bandwidth to the Tor network; that is, the relay could be cheaply instantiated on a shared cloud provider or other low-cost hosting service.

Attacker Goals: The goal of the attacker is to disrupt either (i) the Tor network in its entirety or (ii) a portion of it that affects an entire subpopulation of Tor users. The latter includes attacks against Tor's *bridge* infrastructure, the set of unpublished relays that permit the participation of users who are otherwise prevented from accessing the Tor network directly (e.g., due to censorship).

In general, we consider an attack successful if it entirely prevents users from accessing Tor or if it degrades performance to such an extent that the anonymity service becomes too burdensome to use. The latter is of course subjective, but informally, we set a high threshold for what we consider unusable performance. We also note that even in the current (non-attacked) Tor network, its slow performance is already perceived as an impediment to its more widespread use [13]. Degrading performance much beyond Tor's current levels may cause many users to abandon the network.

Attacker Costs: One of our goals is to estimate the monetary cost of performing various bandwidth DoS attacks against different elements of Tor infrastructure. To estimate such costs to the attacker, we build a cost model from publicly available information on pricing of various online stresser services and dedicated hosting services.

3.1 Stresser Services

There is an active online market for stresser (also called DoS-for-hire or booter) services. These provide the capability to launch DoS attacks against any target, using a web-based interface, at a relatively low monthly cost. Most commonly, the attacks use a distributed botnet of compromised hosts to target a single victim, flooding it with requests.

We summarize the stresser service landscape in Table 1, although this table does not likely capture all available stresser sites. (Since such services are illegal in most jurisdictions, they are not widely advertised and there is significant churn in the industry, making it difficult to obtain a comprehensive list.) We emphasize that Table 1 reports *advertised* attack strengths. Although others have empirically evaluated the achieved attack strengths of these services [74, 75], we elected not to repeat their experiments due to ethical concerns (specifically, the strong possibility of incurring collateral damage). Previous work has found them

Table 1: The estimated mean hourly cost to flood a single target with 1 Gbit/s using various online stresser services. The amortized cost is the hourly price per Gbit/s of traffic per target.

Stresser Service	Time (hrs)	Num Attks	Strength (Gbit/s)	$/mo. (USD)	$/target/hr. (USD)	Amort. (USD)
bootyou.net	3	3	45-50	$ 40	$ 4.44	$ 0.10
booter.xyz	1.67	1	150-200	$ 50	$ 30	$ 0.20
str3ssed.me	1	1	250	$ 55	$ 55	$ 0.22
cloudstress.com	1	1	750	$ 55	$ 55	$ 0.07
ragebooter.net	2	3	10+	$ 60	$ 10	$ 1.00
critical-boot.com	1	1	8-12	$ 40	$ 40	$ 5.00
fiberstresser.com	1	1	750	$ 55	$ 55	$ 0.07
netstress.org	0.67	1	320	$ 45	$ 68.25	$ 0.21
quantumbooter.net	1	2	50	$ 60	$ 30	$ 0.60
vbooter.org	1	3	48-64	$ 40	$ 13.33	$ 0.28
iddos.net	2	1	50	$ 50	$ 25	$ 0.50
downthem.org	0.22	2	200	$ 60	$ 135	$ 0.68
					Mean amortized cost ($/target/hour/Gbit/s):	**$ 0.74**

Table 2: The estimated mean hourly cost to flood a single target with 1 Gbit/s using various dedicated server providers. The amortized cost is the hourly price per Gbit/s of traffic. Prices include 4 CPU cores with minimum 16 GB RAM and 500 GB storage.

Service	Speed (Gbit/s)	Quota (TB)	$/mo. (USD)	Amort. (USD)
Liquid Web	1.00	5	$ 249.00	$ 0.35
InMotion	1.00	10	$ 166.59	$ 0.23
DreamHost	Unkn.	Unmet.	$ 249.00	–
GoDaddy	1.00	Unmet.	$ 239.99	$ 0.33
BlueHost	0.10	15	$ 249.99	$ 3.47
1&1	1.00	Unmet.	$ 130.00	$ 0.18
FatCow	Unkn.	15	$ 239.99	–
OVH	0.50	Unmet.	$ 119.99	$ 0.33
SiteGround	1.00	10	$ 269.00	$ 0.37
YesUpHost	1.00	100	$ 249.00	$ 0.35
			Mean amortized cost ($/hour/Gbit/s):	**$ 0.70**

capable of launching bandwidth DoS attacks with measured attack rates in hundreds of Gigabits per second [74, 75] .

In our analysis in subsequent sections, we consider the average amortized cost of the attacker to flood a single target (i.e., IP address) with 1 Gbit/s of attack traffic for an hour; we found this to be $0.74. Although at first blush, this may appear unrealistically inexpensive, we note that this is *more* costly than obtaining the equivalent bandwidth from legitimate dedicated hosting providers; see §3.2. In §4, we evaluate the cost of using stresser services to overwhelm the capacity of the Tor network en masse, and consider more efficient and targeted stresser attacks in §5 and §6.

3.2 Dedicated Server Costs

Online dedicated hosting services provide customers with remotely located and managed physical machines. Users have full access to the provisioned resources such as CPU, RAM, and disk storage, at a fixed cost. Typically, providers impose some limits on the amount of monthly traffic, and charge different rates for the provisioned network bandwidth. Table 2 reports the pricing schemes for several popular dedicated hosting services. The average amortized hourly cost for transferring data at 1 Gbit/s is $0.70.

Unlike stresser services, dedicated hosting services do not cater to network attackers. They are much more likely to police their traffic and terminate service for customers who are obviously attempting to perform flooding attacks. We thus do not consider their use for naïve flooding of Tor components (e.g., to overwhelm their capacity).

Dedicated servers are well-suited for an attacker who is looking to disrupt the Tor network by leveraging some aspects of Tor's design or protocols. The attacker can use the resources provided by dedicated hosting services to launch such application layer bandwidth DoS attacks on Tor. We explore how dedicated hosting services could serve as a platform for causing severe congestion of Tor relays in §7.

4 Naïve Flooding Attacks against Tor Relays

A straightforward method of attacking Tor is to flood relays with spurious traffic. In this section, we analyze the cost of using stresser services to disrupt the entire Tor network.

Saturating Links: To simplify our analysis, we assume a model of the Internet in which every node i has a finite bandwidth capacity $\mathscr{C}_i$, measured in bits per second (bit/s). We do not consider asymmetric bandwidth since Tor relays receive and send traffic in roughly equal proportions; if node i has asymmetric connectivity, we can consider $\mathscr{C}_i$ to be the minimum of its upstream and downstream capacities.

We assume that an adversary can effectively deny service to a targeted node v if (i) it can cause traffic to arrive at the target at a rate greater than $\mathscr{C}_v$ and (ii) such traffic cannot be filtered upstream. Importantly, the second criterion requires the attacker to initiate a distributed DoS attack from multiple sources (i.e., IP addresses) that cannot easily be enumerated or blocked. Additionally, the communication should resemble legitimate traffic (e.g., be directed at a relevant TCP port). Stresser services generally meet these requirements.

Our first assumption—i.e., a node v effectively becomes unusable if it receives attack traffic at a rate greater than $\mathscr{C}_v$—is admittedly an oversimplification. However, we speculate that saturating v's link would induce a high packet loss rate of 50% or more for legitimate clients, since such clients would have to compete for v's connectivity. TCP performs poorly at such high packet loss rates [61, 68].[1] Stresser services offer attack rates that *vastly* exceed the estimated bandwidth capacities of Tor relays.

Estimating Tor Relay Link Capacity: For a successful flooding attack, the rate at which the attack traffic arrives at

[1] Computing TCP's performance for a given packet loss rate is complex since there are a variety of TCP congestion control algorithms (e.g., Tahoe, Reno, etc.). However, we can derive the theoretical network limit based on the Mathis et al. [61] formula: assuming an average RTT of 40ms, an MSS of 1460B, and a 50% loss rate, the maximum possible throughput achievable by TCP is just $(\mathrm{MSS}/\mathrm{RTT}) \cdot (1/\sqrt{\mathrm{loss}}) = 0.41$ MiB/s.

the target should be equal to or greater than the target's network link capacity. Importantly, we distinguish between the link capacity $\mathscr{C}_v$ of a victim relay and its effective throughput, the latter of which depends on rate limiting, its selection probability, etc. The flooding attack instead depends on overwhelming the victim's actual connectivity, i.e., $\mathscr{C}_v$.

Unfortunately, Tor relays do not publish their link capacities. To estimate a given relay's link capacity, we consider its bandwidth history as recorded in the previous year (from `2017-11-01` to `2018-11-01`) by the Tor Metrics Portal [11]. For each day, we find the maximum observed bandwidth for the relay and map this bandwidth to the next highest value in a fixed set of *bandwidth offerings* that are commonly available: 1, 10, 100, 200, 500, 1,000 and 10,000 Mbit/s. For example, a relay with a maximum observed bandwidth of 1,200 Mbit/s will be considered to have 10 Gbit/s network link. We thus assume that an attacker must direct 10 Gbit/s of attack traffic to overwhelm the relay's capacity. We again emphasize that this is an estimate; the actual capacity at any given time may vary significantly if relay operators configure Tor bandwidth limitation options (operators can set instantaneous bandwidth rate limits and total monthly usage limits).

Attack Cost: We estimate that the total link capacity across the Tor network ranged from 429 to 575 Gbit/s over the year; for our analysis, we use the average of 512.73 Gbit/s. We require that at least one stresser account be used for each Tor relay (since stresser services usually restrict the number of targets to one). Additional stresser accounts are needed to saturate relays with high bandwidth capacities. Applying our cost model, an attacker can use stresser services to flood *all* relays in the Tor network at a cost of about $10K/hr. (or $7.2M/mo.). An adversary can roughly halve its costs by targeting only exit relays, which are required for traffic exiting the network. Overall, however, we find that disrupting Tor by renting stresser services is an expensive proposition, only potentially viable for a nation-state adversary.

Limitations: A limitation of our analysis is that it is not based on empirical evidence (since we were not willing to use such services) and relies on advertised attack rates. Although Santanna et al. have found such services to reasonably deliver high-bandwidth [74, 75], it is possible that they provide a much lower attack strength than advertised. We also rely on the assumption that packets are not filtered upstream, which may not always be valid (as discussed below).

Mitigation: It is possible that ISPs could render such attacks ineffective by filtering traffic. For example, ISPs could discover hosts belonging to the stresser services and filter traffic originating at those hosts. However, such rules may be difficult to maintain given the dynamic nature of the Internet.

Filtering all incoming requests to Tor relays that do not originate at other relays would be an ineffective strategy. In particular, entry relays must allow for clients anywhere on the Internet to initiate a circuit, and any relay may be chosen as an entry by clients implementing non-default path selection algorithms. Filtering attempts may also be complicated by the churn rate of Tor relays and would interfere with the process of bootstrapping new relays to the network. Finally, dropping packets on the relay is an ineffective defense since the dropped packets have already consumed bandwidth.

Traditional DoS defenses such as the use of CDNs are not compatible with Tor since aggregating relays onto a small number of CDN providers would diminish anonymity; it is also unclear how a relay could operate within a CDN. Relay operators may consider migrating to popular cloud services that offer DoS protection services [1, 2]. However, the security and privacy implications of migrating to such services is unknown and may risk exposure to traffic correlation attacks.

Perhaps the most tractable mitigation strategy is to increase the total relay bandwidth capacity of the network.

5 Congesting Tor Bridges

Tor provides anonymous communication to clients, but does not conceal the network locations of its relays, subjecting them to trivial blocking. To counter censors that block access to Tor relays, Tor logically separates *anonymity* (accessing the Internet without revealing network location) from *unblockability* (gaining access to the Tor network). The latter is achieved through the use of *bridge* relays that are not published in the Tor directories. Bridges serve as alternative ingress points into the Tor network for users who cannot directly connect to Tor entry guards.

In this section, we explore the effects of using stresser services (§3.1) to flood Tor bridge relays. We differentiate between three classes of bridges:

Default Bridges: The Tor Browser Bundle (TBB) includes a set of 38 hard-coded default bridges (as of version 8.0.3). Users who cannot directly access Tor relays can configure TBB to connect via one of these default bridges.

A special case of default bridges is *meek bridges* [4, 6] that reside on popular cloud providers and communicate with Tor clients via HTTPS. Censors cannot easily distinguish meek traffic from more typical HTTPS traffic entering the cloud. Disrupting meek thus entails entirely blocking access to the cloud provider, which is presumed to impose too high a collateral cost to the censor. Meek bridges, however, are expensive to operate (since cloud services are not free) and are susceptible to cloud providers disallowing their use [17, 33].

Unlisted Bridges: Users can also request an unlisted bridge either directly from TBB, via bridges.torproject.org, or through email. To prevent a censor from trivially enumerating the bridges, Tor limits the amount of bridges it disseminates to a single requesting IP or email address. However, such protections are obviously brittle and numerous techniques exist for discovering unlisted bridges [20, 30].

Private Bridges: Finally, private bridges are not disseminated by the Tor Project, either because their operators did not notify the Tor Project that they exist or because the Tor Project opted not to disseminate them.

5.1 The State of Tor's Bridges

We first examine the performance of the network's bridges. We focus on the 25 default bridges that use the obfs4 obfuscation protocol[2] since 90% of all bridge users use default bridges [62] and obfs4 is the bridge type recommended by Tor. To test their performance, we use a modified version of Tor to download a 6 MiB file through each bridge. Surprisingly, we find that only 48% (12/25) of the obfs4 default bridges included in TBB are operational.

Figure 1 plots the cumulative distribution (y-axis) of the throughput of the functioning obfs4 default bridges (blue line) when downloading a 6 MiB file on 2018-04-10. Each default obfs4 bridge downloaded the 6 MiB file three times; the CDF plots the average of these downloads. For consistency, we fixed the middle and exit relays, choosing relays with high selection probabilities (and thus high bandwidths). The median throughput of the default bridges is 368 KiB/s; there is a large variation over the default bridges however, ranging from 67 KiB/s to 1,190 KiB/s.

To compare against the performance of unlisted bridges, we requested 135 unlisted obfs4 bridges from the Tor Project's bridge authority via its web and email interfaces. Roughly 70% (95/135) of the acquired unlisted bridges were found to be functional. As shown in Figure 1 (orange line), the unlisted bridges generally outperformed the default bridges, which is expected given Matic et al.'s finding [62] that suggests approximately 90% of bridge traffic is conducted through default bridges. We suspect that the high demand on the few operational default bridges leads to worse performance than the less frequently used unlisted bridges.

As a point of comparison, historical data from the Tor Metrics Portal reveals that non-bridge circuits on Tor during the same time period yielded an average throughput of 786 KiB/s and experienced negligible failure rates [11].

In summary, Tor's bridges are generally far more brittle compared to the network's advertised relays, offering much greater failure rates for default (52%) and unlisted (30%) bridges (compared to 0% for Tor relays) and lower average throughput (545 KiB/s and 681 KiB/s for default and unlisted bridges, respectively, versus 786 KiB/s without bridges).

5.2 Attacking Default Bridges

Ninety percent of bridge users use default bridges [62], and only 12 working default obfs4 bridges are included in the TBB. We first estimate how costly it would be for an attacker to disrupt all of the default bridges. Then, for various migration models in which some percentage of affected bridge users switch to unlisted bridges, we estimate the performance and pricing effects of the migration.

Denying Access to the Default Bridges: Since bridge relays do not publish their bandwidth capacities, our analysis assumes that the distribution of link capacities for n default bridge relays is the same as the distribution of link capacities for the fastest n non-bridge Tor relays. Thus, saturating one default bridge's Internet connectivity requires an amount of bandwidth equal to the link capacity of the fastest Tor relay, and saturating 10 default bridges' requires bandwidth equal to the combined link capacity of the fastest 10 Tor relays. Following the link capacity estimates based on bandwidth offerings as described in §4, we estimate that the set of 12 operational default bridges consists of two 10 Gbit/s links and ten 1 Gbit/s links (a total of 30 Gbit/s) and that the full set of 38 default bridges consists of two 10 Gbit/s links and thirty-six 1 Gbit/s links (a total of 56 Gbit/s). Recall from the pricing model in §3.1 that a 1 Gbit/s stresser account costs $0.74/hr. Attacking the 12 operational obfs4 bridges thus requires 30 of such stresser accounts at a cost of $0.74·30=$22.20 for each hour of downtime (or roughly $22.20·24·31≈$17K per month). Repairing the remaining default bridges offers only a small improvement: denying service to 38 bridges requires 56 stresser accounts at a cost of $0.74·56=$41.44/hr. ($41.44·24·31≈$31K/mo.) which we posit is well within the budget of a nation-state adversary. We emphasize that these are estimates since bridges' true link capacities are unknown.

If the default bridges are successfully attacked, there are several potential consequences. In the worst case, the set of default bridges will not be updated and the users who had depended on them will abandon Tor altogether. The Tor Project could also update its list of default bridges (e.g., by pushing an update to TBB), but such a solution is only temporary since an attacker could simply retarget its DoS efforts.

Users who are dependent on bridges may switch to using either unlisted bridges (since they are more plentiful and more difficult to enumerate) or to meek bridges.

The Cost of Migrating to Unlisted Bridges: We base our analysis on (i) the distribution of throughput we measure from unlisted bridges (Figure 1), (ii) the simplifying assumption that how Tor is used by bridge users is independent of the particular type of bridge used to gain entry to the network, and (iii) Matic et al.'s observation that suggests approximately 90% of bridge traffic traverses through default bridges [62]. If all default bridge users switched to unlisted bridges, applying our simplifying assumption, we would therefore expect the load on the unlisted bridges to increase by a factor of nine (since they previously carried just 10% of bridge traffic). More generally, when a fraction f of default bridge users shift to using unlisted bridges, the unlisted bridges should expect to see a corresponding increase in traffic of a factor of $9 \cdot f$. This trend is plotted in Figure 2.

Given that most (90%) of bridge traffic that is handled by the default bridges, even a small migration of default bridge traffic to unlisted bridges has performance consequences. Even if a quarter of previously default bridge users switch to unlisted bridges, their performance will significantly suffer, decreasing from 762 KiB/s to 338 KiB/s in the median.

[2] obfs4 obfuscates Tor traffic to appear as a random sequence of bytes, making it hard for DPI systems to classify.

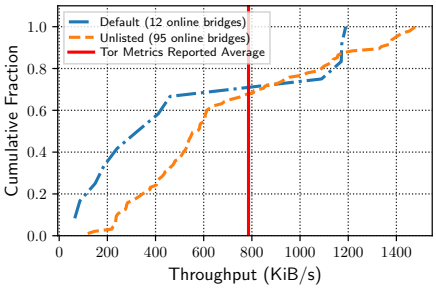


Figure 1: Cumulative distribution of bridge throughput when downloading 6 MiB files. The vertical line at 786 KiB/s shows the throughput for clients that directly connect to Tor to download a 5 MiB file.

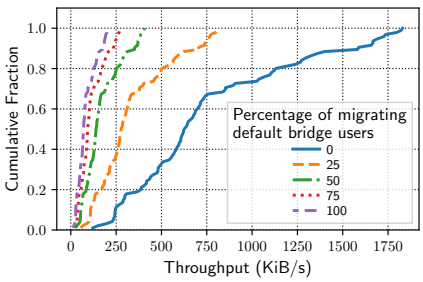


Figure 2: Throughput of Tor users who use unlisted bridges, as a function of the percentage of default bridge users who switch to using unlisted bridges.

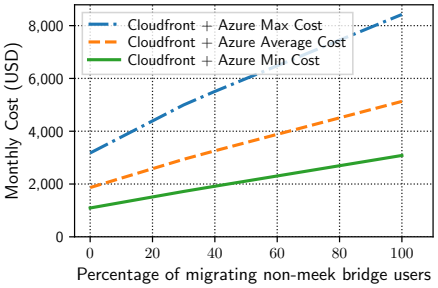


Figure 3: The minimum, maximum, and average cost of maintaining meek when some fraction of users switch to meek, based on meek usage data and CloudFront and Azure pricing models.

The Cost of Switching to Meek Bridges: If the non-meek default bridges become unavailable, we expect some fraction of users to switch to meek bridges. Censors cannot easily disrupt meek bridges without inflicting significant collateral damage since it cannot easily distinguish meek traffic from more typical HTTPS traffic that accesses the cloud provider. However, the increased use of meek incurs a cost (since cloud services are not free).

To estimate the resulting monthly cost of maintaining meek bridges as non-meek users switch over, we estimate the bandwidth consumption of migrating users by constructing a regression model. The bandwidth consumption estimate is required to calculate the cost of operating the meek frontend as cloud providers charge their users based on the bandwidth consumption. To construct the regression model, we first use the statistics from the Tor Metrics Portal to estimate (i) the number of meek users and (ii) the traffic transferred by the meek frontend as reported by meek frontend operators [5] over this same timespan. We use this regression model to estimate the consumed bandwidth as a function of increased traffic from users migrating to meek bridges. We then use the estimated bandwidth usage to derive the expected meek operational cost by applying the current pricing model of the cloud service providers. Unfortunately, the Tor Project stopped providing usage statistics for meek bridges, so we restrict our analysis to usage that occurred between March 2016 and March 2017 (where such data is available [5]). We also note that Amazon and Google stopped supporting the use of domain fronting on their respective cloud services [17, 33][3]. Given these limitations, our analysis represents a rough approximation.

Figure 3 shows the monthly cost of operating meek bridges as a function of the fraction of non-meek bridge users who switch to using meek bridges. We plot the ranges of estimated monthly costs, since cloud providers charge different amounts based on the locations of clients. We note that if half of non-meek users begin using meek bridges, then *in the best case*, the operational cost of maintaining meek bridges will double. If clients are disproportionately in locations in which providers charge higher rates, then the operational costs could be as high as six times the current amount.

Mitigation: Meek bridges offer the best protection against DoS. Unfortunately, supporting a large user base is expensive. One potential strategy for reducing costs is to require bridge users to watch ads or perform small tasks (akin to Mechanical Turk) to finance the cost of their bridge use. Additionally, the recent proliferation of encrypted Server Name Identification (SNI) parameters [36, 44] may enable new methods of domain fronting [32] that are compatible with lower-cost hosting providers.

[3]This highlights a particular brittle aspect of meek bridges—they are completely dependent upon the cloud service on which they reside.

6 Unbalancing Load

In this section, we seek to better understand the extent to which an adversary can disrupt Tor by using stresser services (§3.1) to launch bandwidth DoS attacks on TorFlow [72], a critical component in Tor's load balancing process.

6.1 Relay Performance and Path Selection

As of `2018-11-01`, the Tor network contains 6,436 voluntarily operated relays, the status of which is maintained by 9 Tor *directory authorities* in a signed *network consensus document*. When using Tor, clients download and verify a recent consensus, and use it to select paths of relays through which they build *circuits* and tunnel Internet connections.

Tor uses a load-balancing system in order to provide low-latency anonymous communication (i.e., suitable for browsing websites) due to high client resource demand and a large variance in the bandwidth capacities offered by relays (see §4). The load balancing system is composed of two primary components: a relay performance estimation mechanism and a performance-aware path selection algorithm.

Relay Performance Estimation: Although Tor initially estimated relay performance according to self-reported *advertised bandwidth* capacities, Bauer et al. showed how a low-resource adversary could attract significant traffic to mali-

cious relays (to improve end-to-end correlation attacks) by lying about their available bandwidth [15]. Perry subsequently designed and published the TorFlow relay measurement system to reduce the extent to which Tor trusts relays to honestly report their bandwidth capacity [72], and Tor has been using it to measure relays for nearly a decade (despite alternative designs [19, 55, 78]).

TorFlow is a measurement tool that scans Tor relays to measure their relative performance. To measure Tor relays, TorFlow (i) sorts the consensus list of relays by their previously-expected performance, (ii) partitions the sorted list into *slices* of 50 relays each, (iii) distributes the slices among 9 subprocesses that run in parallel, (iv) creates 2-hop circuits using pairs of relays that belong to the same slice (and so can provide similar performance), and (v) downloads one of a set of 13 fixed-sized files (2^i for $i \in [4, 16]$ KiB) from a known destination through each circuit. TorFlow repeats this process until it has attempted to download through each relay at least 5 times, after which it uses the mean of the measured download completion times to compute a weight for each relay that represents its performance relative to the other measured relays.

The output of the TorFlow measurement process is a *version 3 bandwidth* (V3BW) file specifying the weights and other information about each scanned relay. Currently, 5 of the 9 directory authorities also act as *bandwidth authorities* [3]: they obtain a V3BW file and participate in a voting protocol to determine an authoritative set of relay weights that will appear in the next network consensus. Note that a bandwidth authority operator may obtain a V3BW file by running TorFlow (potentially on a distinct machine from that which runs their directory authority) or by obtaining one from another trusted source that is running TorFlow.

Performance-Aware Path Selection: Tor's path selection algorithm biases relay selection to favor those providing more resources and better performance. Clients using the default selection algorithm will choose relays roughly proportional to the weights assigned to them (via the bandwidth authority voting procedure) and listed in the consensus. As a result, client traffic will be driven to the better performing relays. Previous work has shown that Tor's relay selection strategy does a reasonable job of balancing load [82].

6.2 Detecting TorFlow Scanners

The TorFlow relay scanners constitute attractive targets for DoS attacks: disrupting the scanners may result in significant variation in relays' weights which could degrade load-balancing and security. If the bandwidth authorities were taken offline, Tor would eventually fall back to an equal weighting (uniformly at random) strategy, which would have a detrimental effect on client performance [82]. Previous work observed the ability to detect TorFlow scanners due to their connection patterns and fixed-size file downloads [55], which we further explore.

TorFlow scanners stand out from normal Tor clients because: (i) they download one of a set of 13 fixed-size files, (ii) they choose new entry relays for each circuit (disabling the guard feature), and (iii) they use two-hop circuits.

To discover the network addresses of the TorFlow scanners, we first determined the range in the number of Tor cells required to download each of the fixed-size files. We then operated a low-bandwidth relay and patched it with a small program that looked for connecting clients (potential candidates for TorFlow scanners) that exhibited similar telltale fetches. To provide some ground truth, we also operated our own TorFlow scanner. Within 48 hours, we were able to identify six IP addresses that fetched files through our relay and fit the pattern of a TorFlow scanner. We operated our TorFlow scanner detection software for 5.5 days, during which it did not identify any additional potential scanners. Although we temporarily stored candidate scanner IP addresses in memory (in order to determine uniqueness), we did not write them to stdout or the filesystem (in order to avoid accidentally recording the IP of a human Tor user). We did, however, record that one of the six identified candidate TorFlow scanners was indeed our own; we posit that the other five correspond to the five scanners operated by the Tor Project.

6.3 Attacking TorFlow Scanners

Given that an adversary can identify TorFlow scanners by their IP address, they can use bandwidth DoS attacks to disrupt the relay scanning process and therefore degrade the accuracy of the relay weights produced by TorFlow. A bandwidth DoS attack will clog the TorFlow scanners' links, increasing latency and packet loss on those links and extending the time it takes the scanners to successfully complete file downloads through Tor relays. Therefore, the adversary may effectively manipulate the scanner into believing that relays provide worse performance than they can actually provide. Since TorFlow weights relays by their performance, the adversary can effectively reduce the accuracy of the relay weights which may disrupt the load balancing process.

6.3.1 Attack Strategies

We explore several strategies that an adversary may use to conduct bandwidth DoS attacks on TorFlow scanners with a goal of increasing the file download times measured by TorFlow and disrupting the load balancing process. Each strategy will come at a different cost due to the bandwidth required to conduct the attack and the length at which the attack must be sustained.

Constant: The most straightforward strategy is to simply flood each TorFlow scanner with bandwidth at a constant rate over time. This brute-force strategy is the easiest to set up and should require minimal monitoring and maintenance by the adversary throughout the duration of attack.

Periodic: Since TorFlow produces weights that represent relay performance *relative to other relays*, and because a constant attack strategy may similarly affect all relay measure-

ments, a constant strategy may be suboptimal. Therefore, we also consider a periodic strategy where the adversary floods the victim with bandwidth for a duration of time λ while periodically pausing the attack for a duration of time π. The reasoning behind this strategy is that the scanner will measure normal download times for some relays but significantly reduced download times for others, and the large difference will have a greater impact on the final set of relay weights.

Targeted: We also consider a targeted strategy where the adversary carefully selects periods of time during which to run the DoS attack and otherwise does not alter the victim scanner's network conditions. In particular, we observe that the greatest impact in performance will likely result from significantly depressing the relative weights of the best performing relays. Therefore, the adversary targets the scanner with a bandwidth DoS attack while it is measuring the fastest relays. We discuss below how to determine when the fastest relays are being measured.

6.3.2 Attack Strength and Other Assumptions

For any strategy used by the adversary, we assume that it can utilize a stresser service (see §3.1) to limit the victim's effective bandwidth to rate γ while increasing packet loss on the victim's link by ρ. We assume that the adversary can increase or decrease the attack strength to achieve these effects. (See §6.5 for a discussion of cost.) We also assume that the adversary can receive feedback on the attack by closely monitoring the consensus weights and checking how relays' weights are changing over time. It can monitor the Tor metrics website and data to observe changes in Tor performance. It can iteratively adjust the attack strength and strategy over time in an attempt to produce a greater effect. We also assume that the adversary is capable of setting up and running its own TorFlow scanner instance (the code is open-source), and use it to directly observe how an ongoing attack is affecting the TorFlow measurements and outputs.

The *Targeted* attack strategy depends on being able to target the slice containing the fastest relays. We speculate that the adversary would be able to detect when the fastest slice is being measured by running a fast relay itself and observing when its relay is first measured by a TorFlow scanner. Once detected, the adversary could enable the attack for the time required to measure the slice, which it could estimate empirically by running a TorFlow scanner itself and observing the times to measure the fastest slice over several scan periods. (We ran a TorFlow instance, analyzed its output, and computed the time to measure the fastest slice over 20 scans. We found that the median time to scan the fastest slice was 249 minutes, with an interquartile range of 73 minutes.) Note that these techniques would require additional time, bandwidth, and skill compared to a brute-force attack, and that scan times may be inconsistent over time and network location. See §6.6 for further discussion.

6.4 Evaluation

We evaluated the DoS attack strategies and effects in Shadow [47], a high-fidelity network simulation framework that directly executes Tor. We used Shadow to create a private Tor network that is completely contained inside of our lab environment in order to guarantee that our attacks do not harm the safety or privacy of real Tor users or the network. All of the experiments that we present in this section use Shadow `v1.13.0` and Tor `v0.3.0.10`.

Network Setup: We used standard Shadow and Tor network generation tools and methods [48] to generate a private Tor network with 100 Tor relays, 3,000 Tor clients, and 1,000 server, and to generate background traffic [53]. 2,619 of the clients are *web* clients that download a 320 KiB file, "think" by pausing for a time selected uniformly at random in the range [1,60] seconds, and then repeat. 81 of the clients are *bulk* clients that repeatedly download a 5 MiB file without pausing between successive downloads. We also run 300 *benchmark* clients that reproduce Tor's performance benchmarks by occasionally downloading 50 KiB, 1 MiB, and 5 MiB files using fresh circuits throughout each experiment. We use the most recently published Shadow network topology graph [53] to model inter-host latency.

We implemented a TorFlow plugin for Shadow by significantly refactoring and extending previous work [55]. We used the plugin to scan the relays in our network and produce V3BW files which were then added to the consensus and used by the clients to build paths. We first ran one longer experiment allowing TorFlow time to scan through all relays several times, and then we used the final V3BW file that TorFlow produced as the starting point for all other experiments.

Parameter Settings: We simulated a bandwidth attack by adjusting TorFlow's available bandwidth γ and added packet loss ρ. During each phase where the attack is active, we limit TorFlow's bandwidth to $\gamma = 500$ Kbit/s (62.5 KiB/s) and we add a $\rho = 2\%$ chance of packet loss occurring independently on all incoming and outgoing packets. We set our TorFlow instance to conduct 4 parallel *probes* (2-hop relay measurements), to partition the relays into 10 slices of 10 relays each, and to probe each relay at least 3 times per round before producing a new V3BW file.

We ran a baseline *No Attack* experiment and experiments with each attack strategy. When running the *Constant* attack strategy, the attack is active (the γ and ρ rates applied) for the duration of the experiment. In the *Periodic* attack strategy, the attack cycles through an active period lasting $\lambda = 60$ seconds and an inactive period lasting $\pi = 20$ seconds. In the *Targeted* attack strategy, the attack is active while relays in the slice containing the fastest guard relay in the network are being measured, and inactive otherwise.

TorFlow Scanner Performance: The performance of the TorFlow measurement probe downloads across our experiments is shown in Table 3. As shown in the table, our results indicate that the *Constant* attack is the most effective at caus-

Table 3: The failure rate of TorFlow probe downloads, and the mean (± standard deviation) download rate for each TorFlow probe download and time to complete a full network scan.

Strategy	Fail Rate	Download Rate	Scan Time
No Attack	6.0%	390±381 KiB/s	47±21 min.
Periodic	8.6%	256±292 KiB/s	59±20 min.
Targeted	14%	275±293 KiB/s	80±19 min.
Constant	22%	8.7±5.1 KiB/s	173* min.

*Only a single scan completed in our 300 minute simulation.

ing TorFlow download errors (which increased to 22% from 6% with *No Attack*). The *Constant* attack is also the most effective at limiting the probe download rate, achieving a reduction in mean download rate of about 381 KiB/s, and intuitively increasing the time to scan all relays in the network by about 126 minutes. We find that the *Periodic* and *Targeted* strategies are less effective than the *Constant* strategy, but still do have a measurable effect on the scanner.

Relay Performance: Figure 4(a) shows the relay performance in terms of the distribution of total relay goodput (summed across all relays in the network) over every second during the simulation. We notice a similar trend as with TorFlow performance: in the medians, total relay goodput drops by 86 MiB/s (56%) from 153 MiB/s with *No Attack* to 67 MiB/s with the *Constant* strategy, and relay utilization gets progressively lower with the *Periodic*, *Targeted*, and *Constant* strategies, respectively. Such significant drops in throughput indicates that the new weights produced by TorFlow during the attacks no longer do a good job of balancing client load across relays, and the network is less capable of utilizing its available bandwidth resources.

Client Performance: The effects of our attacks on the mean download rate (during active downloads) per client are shown in Figure 4(b). Every attack has a significant effect, with the mean download rate of the median client being reduced by 45 KiB/s from 56 KiB/s with *No Attack* to 11 KiB/s with the *Constant* attack. Client performance also suffers in terms of the download failure rate per client as shown in Figure 4(c): the failure rate for the median client increases by about 23% from about 3% with *No Attack* to about 26% with the *Constant* attack.

Overall, our results show the extent to which an adversary may disrupt Tor performance using straightforward DoS attacks on easy-to-detect TorFlow scanners, and that the simplest constant attack strategy was the most effective.

6.5 Attack Cost

We assume that our TorFlow DoS attacks could be launched using a stresser service. In §3.1 we describe that the amortized cost of a stresser service to provide 1 Gbit/s of attack traffic is $0.74/hr. The *Constant* attack strategy requires that we constantly run the DoS attack on each scanner. Tor runs 5 TorFlow scanners of unknown capacity. If we assume that they all run on 1 Gbit/s links, then the cost to run the DoS attack on all 5 scanners for one month would be $0.74·5·24·31≈$2.8K.

6.6 Discussion

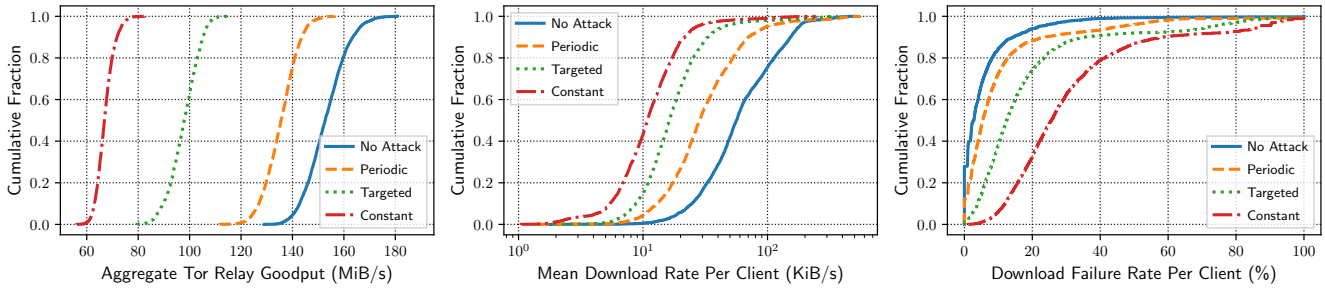
Limitations: A limitation of our study of the effects of bandwidth DoS on the TorFlow scanners is that we used a smaller-scale Tor network than that which is publicly accessible (100 relays compared to 6,436). We used a smaller network primarily due to resource limitations and because TorFlow takes a significant amount of time to scan all relays. Using a smaller network allowed us to (i) run longer experiments, (ii) scan the network faster because there are fewer relays to measure, and (iii) complete more scanning rounds.

Due to scale we configured a single TorFlow instance in our experiments measuring 10 relays per slice, using 4 parallel probe subprocesses, and collecting at least 3 probe measurements per relay; Tor runs 5 TorFlow instances measuring 50 relays per slice while using 9 parallel probe subprocesses and collecting at least 5 probe measurements per relay. The process of partitioning a larger set of relays among more slices and more parallel subprocesses could lead to different inconsistencies than those captured by our simulations. We believe that running additional parallel subprocesses would increase the average bandwidth rate of the TorFlow scanner, which may increase the effectiveness of a constant attack strategy. However, the adversary would require additional bandwidth to attack all or a majority of scanners in parallel.

The *Targeted* attack requires the ability to estimate when the fastest slice is being measured and the amount of time required to measure that slice. These estimates may be complicated by TorFlow's inconsistent relay partitioning and subprocess assignment functions and its parallel measurement processes. In the worst case, the *Targeted* attack would degrade to a *Constant* attack, which performed best in our experiments anyway and for which we estimated cost in §6.5.

The attacks depend on stresser services delivering a high rate of traffic to the target, which is not a stealthy operation. This could potentially trigger automated or manual DoS mitigation techniques, and we did not consider how such defenses would affect the attacks. We rely on the assumption that packets are not filtered upstream from the target scanner, which may not always be valid (as we will discuss below).

Attack Extension: Our focus in this paper is on relatively simple and straightforward bandwidth-based DoS attacks. However, it is possible to extend the attack if we consider a more powerful adversary. For example, a network-level adversary that can observe connections from a TorFlow scanner can selectively disrupt TorFlow as it is scanning a target set of relay IP addresses. This would allow an adversary to selectively increase the time to scan the target set of relays, causing the scanner to detect that those relays are "overloaded" and reduce their weights (and therefore the probability that those relays are used by clients) accordingly. Such an attack could be used to drive additional traffic to other mali-



(a) Relay Utilization **(b)** Client Performance **(c)** Client Error Rate

Figure 4: Performance metrics as measured when the network is not under attack (*No Attack*) and when the bandwidth authorities are attacked using different strategies (*Periodic*, *Targeted*, and *Constant*): (a) the distribution of aggregate Tor relay goodput per second (summed across all relays for every second); (b) the distribution of mean download rates per client; and (c) the distribution of download failure rates per client.

cious relays, improving the ability of an adversary to conduct attacks on anonymity, e.g., traffic correlation attacks [66].

Mitigation: Since the attacks rely on stresser services, the mitigation strategies discussed in §4 also apply here. Specifically, it is possible that ISPs could render attacks against the bandwidth authorities ineffective by filtering traffic en route. For example, ISPs could discover hosts belonging to the stresser services and filter traffic originating at those hosts, or ISPs could install custom rules to filter incoming traffic to the bandwidth authorities since they need only make outgoing connections (in case they are not run alongside Tor relays). However, filtering attempts may be complicated by packet spoofing. (Note that dropping packets on the bandwidth authority is an ineffective defense since the dropped packets already consumed bandwidth.)

Perhaps the best mitigation is to migrate to a decentralized bandwidth measurement system that does not share TorFlow's security problems. For example, TorFlow's centralized scanning approach can be easily detected because it uses fingerprintable traffic signatures from a small set of static IP addresses. While it may be difficult to obfuscate the source, destination, and traffic signature while still providing accurate results [55], a system that utilizes distributed trust may help thwart malicious behavior. For example, a peer-based measurement system run by existing relays would preclude the need for centralized measurement infrastructure that is vulnerable to DoS and could complicate scanner and measurement detection [55, 78].

7 Congesting Tor Relays

In §4, §5, and §6 we evaluated the effects of using stresser services (§3.1) to flood Tor relays, Tor bridges, and Tor bandwidth authorities, respectively. In this section, we move away from stresser services and explore the extent to which an adversary might degrade Tor network performance by using the Tor protocol itself to congest Tor relays. The attack strategies that we discuss in this section utilize dedicated servers (§3.2) and modified Tor clients.

7.1 Relay Usage

All non-bridge Tor relays are publicly known and distributed to Tor clients in network consensus documents in order to facilitate the Tor client path selection and circuit building processes. By default, clients select 3-hop Tor paths and use them to build circuits that are usable for 10 minutes. Clients must use an *exit relay* that allows exiting the Tor network on the desired TCP port as the last relay in their Tor circuits in order to communicate with Internet peers that are not Tor-aware. Additionally, by default, clients use *guard relays* as their entry into the Tor network, and guard or *middle relays* (i.e., non-guard, non-exit relays) in the middle position of their circuits. As discussed in §6, each relay is assigned a weight by the directory authorities corresponding to its performance relative to all other relays as measured by TorFlow [72]. Clients use these weights during path selection to bias their choice of relays toward those providing better performance.

7.2 Abusing Relay Bandwidth

The Tor protocol contains features that offer protocol flexibility, but also allow for abuse. Although exits are required to be used in the last position of circuits exiting Tor, the Tor protocol technically allows any relay to be used in any non-exit position. Additionally, the Tor protocol technically allows circuits containing up to 8 relays. To utilize these features, a client may select its own custom path of relays and build circuits through them either by modifying their Tor client code directly, or by interacting with Tor using the Tor control interface and protocol. Note that building custom circuits is already supported by existing Tor control clients like stem [8].

7.2.1 Attack Strategies

Given the above features, an adversary may conduct a concerted bandwidth consumption DoS attack by building custom circuits and downloading large files through them.

Long Paths: The most basic form of our Tor bandwidth DoS attack makes use of the ability to create 8-hop Tor circuits.

For every byte of data downloaded by a client through such a *long path*, relays in the Tor network will download and upload that byte 8 times in total. This amplification works significantly in favor of the adversary.

Tunneling: Tor previously allowed infinite-length circuits until it was shown that long paths (e.g., 24 hops) could be used to congest Tor relays and deanonymize clients [31]. Although the Tor protocol now restricts circuits to 8 relays in length [27, §5.6], paths of unrestricted length are still technically possible by using multiple Tor clients and tunneling each client's TCP onion connection to its entry relay through another client's circuit[4] [51].

Stop Reading: In order to decrease the cost of downloading large files, the adversary may use a *Stop Reading* strategy. Using this strategy, the adversary first creates a new TCP onion connection to the first-hop relay in its chosen long path, even if a connection to that relay already exists. The adversary then builds an attack circuit using its chosen path and waits for it to complete successfully, making sure to assign the circuit to the new TCP connection. Once the circuit is built, the adversary sends a request for a large data blob from some public server (e.g., the Internet Archive), measures the time to download the first 25 KiB, and then instructs Tor to stop reading from the circuit's TCP connection to the entry relay (immediately after receiving 25 KiB). Although the client stopped reading, it can still write: the adversary uses the measured time to download the first 25 KiB to estimate the frequency with which it should send Tor circuit and stream `SENDME` flow control cells[5] which instruct the exit relay to continue to send data toward the client. (Estimating the circuit throughput rate is important, because sending more `SENDME` cells than is expected by the exit will cause the exit to abort the circuit.)

A stop reading strategy was first described and used as part of the Sniper Attack [51], but we are the first to observe that each attack circuit should use a new and unique TCP connection in order to limit interference with other circuits built in parallel. This requires minor modifications to the Tor client code since Tor by default multiplexes circuits over existing TCP connections. We show in §7.3 that our bandwidth DoS attack scales to thousands of circuits using this approach.

7.2.2 Attack Targets

Single Relay: An adversary may use an 8-hop long path to conduct a congestion attack on a target victim relay by including the victim in the same circuit multiple times. Since an honest relay will not extend a circuit to the same relay that extended to it, the victim v must be placed in circuit positions such that two distinct honest relays h_1 and h_2 are placed in subsequent positions before repeating the victim (i.e., $v \leftrightsquigarrow h_1 \leftrightsquigarrow h_2 \leftrightsquigarrow v$, etc.). Therefore, a victim may appear in the same circuit a maximum of 3 times (either in positions 1, 4, and 7, or 2, 5, and 8).

Relay Subgroups: An adversary may also target specific subgroups of relays that represent particularly attractive targets. Such subgroups may include the group of all exit relays (since their resources are the most scarce), the group of all publicly known bridges (we explored the impact of such an attack in §5), hidden service directories, and the group of 9 directory authorities (which also serve as relays).

All Relays: An adversary may also attempt to congest the entire Tor network with the goal of degrading performance to the extent that Tor becomes unusable to a majority of its user base, which would significantly reduce Tor's security in addition to its performance [25]. In order to congest all relays, the adversary makes a weighted selection of relays following the weights published in the network consensus. In other words, the adversary uses the same path selection policy as honest clients do by default. This will ensure that the adversary will choose and congest relays with the same distribution that clients attempt to use them: the DoS attack will cause more congestion on relays that are chosen more often by clients, and should therefore impact more users.

7.2.3 Attack Strength

The strength of our attack can be described in terms of the number of long path circuits ϕ that the adversary builds in parallel. Each circuit should use at least 2 parallel streams (i.e., downloads) in order to fully utilize the circuit flow control mechanism (the circuit window is 1,000 cells, twice that of the stream window). Whenever circuits close or downloads finish (complete or time out), circuits are replaced with new ones in order to maintain the attack strength over time.

7.3 Evaluation

As in §6.4, we use Shadow [47] to safely measure the effects of our DoS attacks in a private Tor network. All experiments in this section use Shadow `v1.13.0` and Tor `v0.3.1.10`.

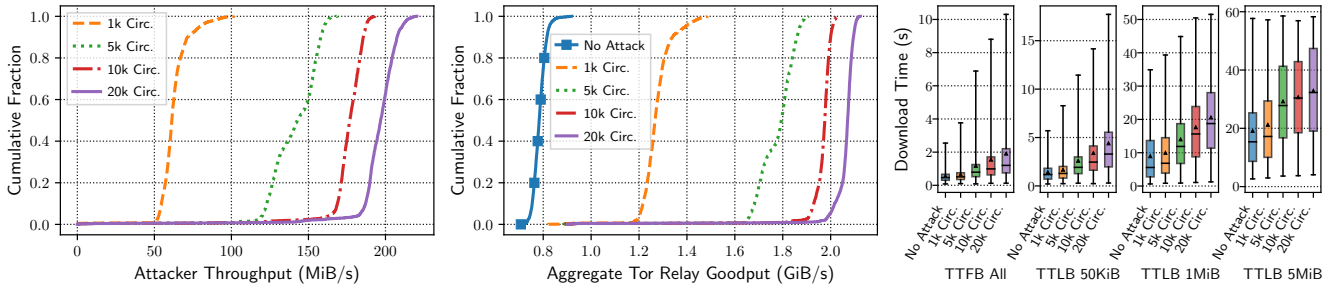
Network Setup: We use the same tools and methods as described in §6.4 to generate a Tor network containing 634 relays (10% of the size and capacity of the public network), 15,000 clients (14,259 *web* clients, 441 *bulk* clients, and 300 *benchmark* clients), and 2,000 servers. Node behaviors are also as described in §6.4.

We implemented our DoS attacks in a C program containing 3,265 lines of code (LoC) which we compiled as a Shadow plugin. Our attacks utilize a Tor `v0.3.1.10` client that we modified (409 LoC) to support creating new TCP connections for attack circuits as well as commands to stop reading and for sending `SENDME` cells.

Parameter Settings: Throughout our experiments, we explore the effects of the *Long Path* strategy across attack strengths (number of circuits ϕ) on performance. We parallelize our attack by running $\phi/1,000$ identical processes on new attack hosts (i.e., 1,000 circuits per host), each of

[4] Tor will tunnel TCP onion connections through a proxy (e.g., another Tor client) when using the `Socks4Proxy` or `Socks5Proxy` torrc options.

[5] A stream `SENDME` cell is sent for every 50 received cells (25 KiB) and a circuit `SENDME` for every 100 received cells (50 KiB).



(a) Attacker Throughput **(b)** Relay Goodput **(c)** Client Performance

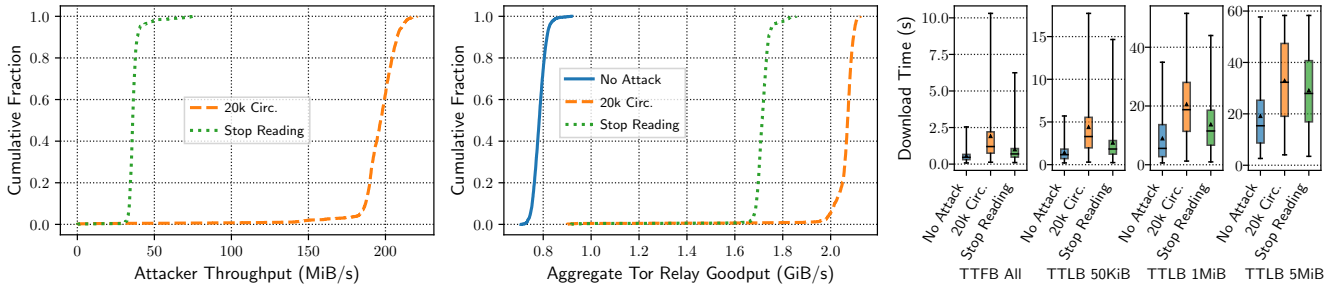
Figure 5: Performance metrics as measured throughout our network-wide DoS attack on Tor relays: (a) the distribution of attacker throughput rates; (b) the distribution of aggregate Tor relay goodput rates (summed over all relays for each second); and (c) the distribution of *benchmark* client download times to first byte (TTFB) and last byte (TTFB) for files of sizes 50 KiB, 1 MiB, and 5 MiB (the box shows the interquartile range, the ▲ shows the mean, and the lower and upper whiskers extend to the minimum and the 99th percentile values, respectively).

which is configured with a 1 Gbit/s network link to ensure that we could measure the full bandwidth cost of the attack. We attach 2 streams that each request 10 MiB of data to each 8-hop attack circuit whose path is chosen depending on the target (we explore a single relay and all relays as targets). We set each attack circuit to time out if any of its streams either have not completed within 5 minutes, or have not received a byte in the most recent 60 seconds. Finally, we run experiments with and without the *Stop Reading* strategy in order to understand the cost and impact of DoS with and without it. (We did not evaluate the *Tunneling* strategy or a strategy that targets *Relay Subgroups*.)

Single Relay Attack: We evaluated the *Long Path* strategy in a *Single Relay* attack against the most highly weighted middle relay in our network. We evaluated attack strengths of 100, 500, and 1,000 circuits. In all cases, the attacker required less than 2.6 MiB/s of throughput to conduct the attack. The victim relay's throughput increased from 978 KiB/s with no attack to 3.8 MiB/s with 100 attack circuits and 5 MiB/s with both 500 and 1,000 attack circuits (in the medians). Interestingly, 500 circuits was enough to consume all of the victim's 5 MiB/s capacity. In the medians, the time to download 1 MiB through the victim increased from 3.3 seconds with no attack to 28 seconds with 1,000 circuits, while the download failure rate increased from 0% with no attack to 63% with 1,000 attack circuits. Our results indicate that the attack has a clear effect on both relay throughput and client performance on a small scale.

All Relays Attack: We evaluated the *Long Path* strategy in an *All Relays* attack using ϕ = 1k, 5k, 10k, and 20k attack circuits. The performance measured during the attacks and compared to a no attack baseline experiment is shown in Figure 5. Figure 5(a) shows the cumulative distribution of the total attacker throughput used during each attack. Intuitively, the throughput used by the adversary increases with the number of attack circuits: from a median of 61 MiB/s in a ϕ=1k circuit attack to a median of 197 MiB/s in a ϕ=20k circuit attack. Note that the increase in attacker throughput is not linear in ϕ, indicating that we may be reaching relay bandwidth resource limits. Figure 5(b) supports this claim, showing the distribution over each second of Tor network goodput (summed over all relays). In the medians, aggregate relay goodput increases from 802 MiB/s with no attack to 1,297 MiB/s for ϕ=1k (an increase of 495 MiB/s) and 2,120 MiB/s for ϕ=20k (an increase of 1,318 MiB/s). Interestingly, we can see from these results that the effect of ϕ=10k circuits is much greater than the effect of doubling the attack strength to ϕ=20k circuits, suggesting diminishing returns. Finally, the effect of our attack on client performance is shown in Figure 5(c) across a range of download time metrics. Generally, the time to complete downloads of various sizes increases significantly with the attack strength (e.g., TTFB increases by 138% and TTLB increases by 120% in the medians across all downloads for ϕ=20k), and the variance in performance also increases as attack circuits are added to the network.

Figure 6 shows the effect of the *Stop Reading* strategy on performance. Compared to the regular *Long Path* strategy in the ϕ=20k attack, Figure 6(a) shows that the *Stop Reading* strategy consumed only 36 MiB/s of attacker bandwidth in the median while Figure 6(b) shows that the attack was still able to consume 1,755 MiB/s of total relay bandwidth in the median (an increase of 953 MiB/s over no attack). Figure 6(c) shows that the effect on client performance is also slightly less pronounced (TTFB increases by 48% and TTLB increases by 47% in the medians across all downloads), which is to be expected given that relays are less congested. We expect that we could further increase congestion by scaling up the *Stop Reading* attack at relatively little bandwidth cost to the attacker. A summary of our results in Table 4 shows that the *Stop Reading* strategy achieved the highest bandwidth amplification factor of 26 primarily due to the reduction in attacker bandwidth usage.



(a) Attacker Throughput **(b)** Relay Goodput **(c)** Client Performance

Figure 6: The effects of the *Stop Reading* attack on the performance metrics as measured throughout our network-wide 20,000 circuit DoS attacks on Tor relays: (a) the distribution of attacker throughput rates; (b) the distribution of aggregate Tor relay goodput rates (summed over all relays for each second); and (c) the distribution of *benchmark* client download times to first byte (TTFB) and last byte (TTFB) for files of sizes 50 KiB, 1 MiB, and 5 MiB (the box shows the interquartile range, the ▲ shows the mean, and the lower and upper whiskers extend to the minimum and the 99th percentile values, respectively).

Table 4: Summary of the total increase in relay bandwidth usage our attack achieved with the given attacker bandwidth usage (all in MiB/s) and the resulting bandwidth amplification factors.

	ϕ=1k	ϕ=5k	ϕ=10k	ϕ=20k	Stop*
Relay Bandwidth	495	1,035	1,221	1,318	953
Attacker Bandwidth	61	143	177	197	36
Amplification Factor	8.1	7.2	6.9	6.7	26

* **Stop** is a ϕ=20k attack using the *Stop Reading* strategy.

7.4 Attack Cost

We assume that our DoS attacks could be launched on a dedicated server. In §3.2 we describe that the amortized cost in the dedicated server model for 1 Gbit/s of traffic is $0.70/hr. (which includes the monthly hardware rental and any bandwidth costs). Our ϕ=20k attack with the *Stop Reading* strategy requires only 288 Mbit/s (36 MiB/s) in our scaled down private Tor network which contains about 10% of both the number of relays in and the bandwidth capacity of the public Tor network (due to the relay sampling approach of Jansen et al. [48]). If we assume that the attack scales linearly with Tor's bandwidth capacity, then our attack would require 10·288 Mbit/s≈3 Gbit/s. The cost to rent three dedicated servers supporting 1 Gbit/s of traffic for one month would then be $0.70·3·24·31≈$1.6K. (The regular version of the attack consumes ≈4 times as much bandwidth, requiring 12 dedicated servers and costing $0.70·12·24·31≈$6.3K/mo.)

Our attack additionally utilizes multiple client IP addresses, each of which is used to maintain an average of 1,000/634≈1.6 circuits per relay. If (i) the public Tor network contains 10·634=6,340 relays, (ii) we create 10·20,000=200,000 circuits, and (iii) we maintain the average 1.6 circuits per relay per client IP address rate, then we would require 200,000/6,340/1.6≈20 client IP addresses. If we assume that one IP address is provided for each of our three dedicated servers, and the cost is $5 per additional IP address, then the additional monthly cost for purchasing 17 more IP addresses is 17·$5=$85.

Thus, we estimate that the total cost to run a ϕ=200k circuit attack using the *Stop Reading* strategy against the public Tor network is still $1.6K/mo. due to rounding. (We estimate that the cost of the regular version of the attack is still $6.3K/mo. due to rounding.)

7.5 Discussion

Limitations: We used Shadow in order to ethically conduct full-network Tor simulations, and simulation inherently incurs some inaccuracy. However, while no simulator is perfect, Shadow has been shown to exhibit network behavior and performance that is very similar to Linux [50, 52]. Additionally, Shadow has been used to measure performance when Tor is generally overloaded (as in our evaluation) [47, 49, 52], and it has been used to measure the effects of specific DoS attacks against Tor [51].

Our experiments are limited in scale. We simulated a private Tor network with about 10% of both the number of relays in and the bandwidth capacity of the public Tor network. The cost to the adversary to conduct our attack may not scale linearly with the amount of Tor capacity as we assumed in §7.4, or there may be other issues that arise when scaling up our attack. We note that we are limited by the capabilities of our tools and resources and highlight that it would be unethical to conduct this work at scale on the public Tor network.

Attack Extensions: We did not evaluate the effects of onion connection tunneling on DoS (i) because Tor could prevent the attack by updating the default exit policy to prevent exiting to a Tor relay, and (ii) in order to provide a more conservative estimate of the bandwidth and monetary costs of performing our bandwidth DoS attack. However, we believe that the technique would be simple to deploy. Additionally, it would be interesting to explore the effects of our attacks on performance when targeting subgroups of relays.

Mitigations: It is extremely challenging to mitigate bandwidth DoS attacks on Tor because the circuits that we build in our attack download an amount of traffic that a reasonable client could realistically download. The *Long Path* part of the attack could be mitigated if Tor changes its protocol to further restrict the length of circuits, however, in this case an adversary could switch to using hidden service circuits which are 6 hops by default.

The *Stop Reading* part of our attack uses a separate TCP connection to the entry relay for each attack circuit, and builds many such connections and circuits in parallel. Tor implemented mitigations to this kind of DoS attack and merged them in early 2018 [24, 39] in response to reports of DoS against relays [23, 38, 40]. In the new subsystem, relays will refuse new TCP connections from any IP address that creates more than 3 concurrent connections, and they will refuse new circuits from the IP address if it *also* creates more than 3 circuits per second with an allowable burst of 90 circuits (these were the default settings on 2018-11-01). Our Tor experiments were configured to run with these DoS mitigations in place, and our attacks did not trigger the DoS defense on any relay. Our attacks were able to stay under the connection threshold because we utilize every relay in the network as an entry relay and we maintain only 1,000 circuits per client IP address (1.6 circuits per relay per client IP address on average). While we do believe that the implemented mitigations are effective against some attacks, we note that the proliferation of IPv6 addressing may further reduce their effectiveness.

A defense against the Sniper Attack [45, 51] was merged in Tor `v0.2.4.14-alpha` (released on 2013-06-13). The defense detects and kills the circuit with the longest waiting cell at the head of the queue if the relay is under memory (RAM) pressure. The defense was active but was not triggered in our experiments since we only download 20 MiB of data through each circuit before abandoning it (our goal is to consume bandwidth rather than a victim's RAM as in the Sniper Attack, so we do not require long queues).

In order to further limit the impact of the *Stop Reading* strategy, we recommend the implementation and deployment of the authenticated `SENDME` design as previously described [51] and specified [46]. With authenticated `SENDME`s, a client would need to continue reading data in order to continue producing authentic `SENDME` cells, and the exit would destroy circuits on which it received invalid `SENDME`s. This defense would limit a stop reading DoS strategy to 1,000 cells (500 KiB) per circuit, effectively mitigating it.

8 Sybil Attacks

We previously explored several bandwidth-based DoS attacks against Tor while estimating the cost to conduct each attack and their effects on Tor performance; we summarize our cost estimates in Table 5. In this section, we compare our DoS attacks with a Sybil attack in which an adversary instead uses its budget to run several high-bandwidth Tor relays in order to affect as much Tor user traffic as possible.

Table 5: A summary of the costs of our main attacks.

Attack (Section)	Service	Bandwidth	Cost
Bridge Congestion (§5)	stresser	30 Gbit/s	$17K/mo.
Load Unbalancing (§6)	stresser	5 Gbit/s	$2.8K/mo.
Relay Congestion (§7)	ded. server	3 Gbit/s	$1.6K/mo.

Table 6: The effective mean aggregate bandwidth resources of Tor relays from `2017-11-01` to `2018-11-01` (in Gbit/s), computed using positional bandwidth weights from `2018-11-01`.

Bandwidth	Entry	Middle	Exit	Total
Usage	42.4 (36.0%)	42.9 (36.4%)	32.5 (27.6%)	118
Capacity	86.7 (35.0%)	96.7 (39.1%)	64.0 (25.9%)	247

Relay Resources: To determine which type of relays would be most advantageous, we computed the effective positional bandwidth usage by and capacity of Tor relays over the year preceding `2018-11-01`. The *effective bandwidth* accounts for relay flags and position weights, both of which are used to determine in which position a relay will be selected. From the results shown in Table 6, we can see that exit bandwidth is the scarcest, with only 27.6% of the total bandwidth used and 25.9% of the total bandwidth capacity.

Sybil DoS Attack: An adversary could run Sybil relays and then arbitrarily degrade the performance of all traffic forwarded through its Sybils, or deny service by dropping circuits. Note that for such an attack to work, the adversary must (i) maintain a high selection probability by providing high performance during periods in which it is measured by Tor's bandwidth measurement system, and (ii) not trigger Tor's abusive relay detection systems (e.g., exit scanners) to avoid getting ejected from the network. We assume that these requirements can be met for the purposes of this analysis.

Due to exit bandwidth scarcity, an adversary can maximize its probability of appearing at least once in a circuit by running all exit relays. We assume that the aggregate bandwidth *usage* (i.e., network load) will remain constant as the adversary adds additional bandwidth *capacity* (i.e., Sybil relays), and that the probability that the adversary serves as the exit in a circuit is approximately equal to its fractional exit capacity (Table 6). Then, Sybil DoS attacks with bandwidth budgets of 30, 5, and 3 Gbit/s (Table 5) could arbitrarily degrade performance for $30/(30+64)\approx 32\%$, $5/(5+64)\approx 7.2\%$, and $3/(3+64)\approx 4.5\%$ of exit circuits, respectively. Comparatively, our attack in §5 affects *all* non-private bridge circuits, and our attacks in §6 and §7 affect *all* circuits.

Sybil Deanonymization Attack: If an adversary is able to observe both the entry and exit points in a circuit (its relays are chosen in the first and last circuit positions), then it is generally assumed that the circuit is *vulnerable* to compromise because traffic correlation can be performed to deanonymize the user with high probability [65, 66]. Note

Table 7: The fraction of circuits affected by Sybil attacks.

Bandwidth	Sybil DoS	Sybil Deanonymization
30 Gbit/s	32% degraded	21% entry · 5.3% exit ≈ 1.1% total
5 Gbit/s	7.2% degraded	4.5% entry · 1.2% exit ≈ 0.06% total
3 Gbit/s	4.5% degraded	2.8% entry · 0.8% exit ≈ 0.02% total

that a selective service refusal attack, where an adversary refuses to forward traffic on any circuit it is not in a position to compromise [16], could be mitigated by Tor's route manipulation (path bias) detection system [26, §7].

In order to observe both ends, an adversary must operate at least one entry guard and at least one exit relay. The entry position is more difficult to obtain since Tor clients use the same guard relay for months at a time [29]. Therefore, previous work has found that a 5:1 guard-to-exit relay bandwidth allocation maximizes the probability of observing both sides of a circuit at least once [54].

A Sybil deanonymization attack with a bandwidth budget of 3 Gbit/s (Table 5) and a 5:1 guard-to-exit relay bandwidth allocation would allow the adversary to observe the entry for $\frac{5}{6}\cdot 3/(\frac{5}{6}\cdot 3+86.7)\approx 2.8\%$ of Tor clients and observe the exit for $\frac{1}{6}\cdot 3/(\frac{1}{6}\cdot 3+64.0)\approx 0.8\%$ of circuits built by those clients. Thus, approximately 0.02% of circuits would be vulnerable. Table 7 shows results for other bandwidth budgets and summarizes our Sybil attack analysis.

Discussion: Note that Sybil attacks require fixed costs, because relays must generally be fast and reliable in order to be properly utilized by the network. Additionally, attacks that require guard relays can take months to observe a full set of *some* clients (due to guard rotation times), and much longer to observe a set containing *specific* clients. Conversely, our attacks are more flexible because they do not require fixed costs, can be run with clients rather than service providers (relays), and can be repeatedly started and stopped as necessary. Further, our attacks immediately affect all clients (rather than some sample), and our relay congestion attack (§7) benefits from the anonymity that Tor provides.

9 Conclusion

This paper performs a multifaceted examination of Tor's vulnerability to DoS, considering both the efficacy of DoS attacks as well as the adversary's cost of performing them. On the positive side, we find that Tor's growth has made it more resilient at least to simple attacks: disrupting the service by naïvely flooding Tor relays using stresser services is an expensive proposition and requires $7.2M/month.

Unfortunately, however, several aspects of Tor's design and rollout make it susceptible to more advanced attacks. We find that Tor's bridge infrastructure is heavily dependent on a small set of fixed default bridges, the operational of which can be disrupted at a cost of $17K/month. Additionally, Tor's mechanism for measuring load is too centralized and brittle, and even inexpensive techniques (e.g., costing $2.8K/month) can significantly perturb these processes and cause dramatic performance degradation across the network. Finally, attackers can saturate Tor's capacity by constructing long paths in the network, and exploit protocol vulnerabilities to decrease the costs of such attacks; for example, we find that an attacker can significantly degrade the performance of the network for as little as $1.6K/month. We also compare our attacks to Sybil attacks and highlight that our load balancing and relay congestion attacks are more effective and flexible than Sybil attacks with the same budget.

For each attack, we describe mitigation strategies that Tor could adopt to improve its resiliency. In particular, we recommend additional financing for meek bridges, moving away from load balancing approaches that rely on centralized scanning, and Tor protocol improvements (in particular, the use of authenticated `SENDME` cells).

Acknowledgments

We thank the anonymous reviewers for their valuable feedback that helped to improve this paper. We thank Nikita Borisov for shepherding our paper and David Goulet for discussions about DoS mitigation in Tor. This work has been partially supported by the Office of Naval Research, the National Science Foundation under grant number CNS-1527401, and the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-16-C-0056. The opinions, findings, and conclusions or recommendations expressed in this work are strictly those of the authors and do not necessarily reflect the official policy or position of any employer or funding agency.

References

[1] AWS Shield Managed DDoS protection. `https://aws.amazon.com/shield/`, November 2018.

[2] Azure DDoS Protection Standard overview. `https://docs.microsoft.com/en-us/azure/virtual-network/ddos-protection-overview`, November 2018.

[3] Consensus Health: Bandwidth Scanner Status. `https://consensus-health.torproject.org/#bwauthstatus`, November 2018.

[4] meek. `https://trac.torproject.org/projects/tor/wiki/doc/meek`, November 2018.

[5] meek Costs. `https://trac.torproject.org/projects/tor/wiki/doc/meek#Costs`, November 2018. Meek Pluggable Transport Frontend Costs.

[6] meek Overview. `https://trac.torproject.org/projects/tor/wiki/doc/meek#Overview`, November 2018. Meek Pluggable Transport Overview.

[7] Online Survival Kit. `https://rsf.org/en/online-survival-kit`, November 2018.

[8] Stem: a Python Controller Library for Tor. `https://stem.torproject.org`, November 2018.

[9] Tor Git Repository Browser. `https://gitweb.torproject.org`, November 2018.

[10] Tor Bug Tracker and Wiki. https://trac.torproject.org, November 2018.

[11] Tor Metrics Portal. https://metrics.torproject.org/, November 2018.

[12] M. AlSabah and I. Goldberg. PCTCP: Per-circuit TCP-over-IPsec Transport for Anonymous Communication Overlay Networks. In *Conference on Computer and Communications Security (CCS)*, 2013.

[13] M. Alsabah and I. Goldberg. Performance and Security Improvements for Tor: A Survey. *ACM Comput. Surv.*, 49(2):32:1–32:36, September 2016.

[14] M. V. Barbera, V. P. Kemerlis, V. Pappas, and A. D. Keromytis. CellFlood: Attacking Tor Onion Routers on the Cheap. In *European Symposium on Research in Computer Security (ESORICS)*, 2013.

[15] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. Low-resource Routing Attacks Against Tor. In *Workshop on Privacy in the Electronic Society (WPES)*, 2007.

[16] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz. Denial of Service or Denial of Security? In *Conference on Computer and Communications Security (CCS)*, 2007.

[17] R. Brandom. Amazon Web Services Starts Blocking Domain-fronting, Following Google's Lead. https://www.theverge.com/2018/4/30/17304782/amazon-domain-fronting-google-discontinued, April 2018. The Verge Online News Article.

[18] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *Conference on Computer and Communications Security (CCS)*, 2012.

[19] H. Darir, H. Sibai, N. Borisov, G. Dullerud, and S. Mitra. TightRope: Towards Optimal Load-balancing of Paths in Anonymous Networks. In *Workshop on Privacy in the Electronic Society (WPES)*, 2018.

[20] R. Dingledine. Research Problems: Ten Ways to Discover Tor Bridges. https://blog.torproject.org/research-problems-ten-ways-discover-tor-bridges, October 2011. Blog Post.

[21] R. Dingledine. Tor security advisory: "relay early" traffic confirmation attack. https://blog.torproject.org/tor-security-advisory-relay-early-traffic-confirmation-attack, July 2014. Blog Post.

[22] R. Dingledine. Did the FBI Pay a University to Attack Tor Users? https://blog.torproject.org/did-fbi-pay-university-attack-tor-users, November 2015. Blog Post.

[23] R. Dingledine. could Tor devs provide an update on DOS attacks? Tor-Relays Email 014175, December 2017. https://lists.torproject.org/pipermail/tor-relays/2018-January/014175.html.

[24] R. Dingledine. Experimental DoS mitigation is in tor master. Tor-Relays Email 014357, January 2018. https://lists.torproject.org/pipermail/tor-relays/2018-January/014357.html.

[25] R. Dingledine and N. Mathewson. Anonymity Loves Company: Usability and the Network Effect. In *Workshop on the Economics of Information Security (WEIS)*, 2006.

[26] R. Dingledine and N. Mathewson. Tor Path Specification. https://gitweb.torproject.org/torspec.git/tree/path-spec.txt, November 2018. Section 7.

[27] R. Dingledine and N. Mathewson. Tor Protocol Specification. https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt, November 2018. Section 5.6.

[28] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*, 2004.

[29] R. Dingledine, N. Hopper, G. Kadianakis, and N. Mathewson. One Fast Guard for Life (or 9 Months). In *Privacy Enhancing Technologies Symposium (PETS)*, 2014.

[30] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-wide Scanning and its Security Applications. In *USENIX Security Symposium*, 2013.

[31] N. S. Evans, R. Dingledine, and C. Grothoff. A Practical Congestion Attack on Tor using Long Paths. In *USENIX Security Symposium*, 2009.

[32] D. Fifield, C. Lan, R. Hynes, P. Wegmann, and V. Paxson. Blocking-resistant Communication through Domain Fronting. In *Privacy Enhancing Technologies Symposium (PETS)*, 2015.

[33] S. Gallagher. Google Disables "Domain Fronting" Capability Used to Evade Censors. https://arstechnica.com/information-technology/2018/04/google-disables-domain-fronting-capability-used-to-evade-censors, April 2018. Ars Technica Online News Article.

[34] J. Geddes, R. Jansen, and N. Hopper. How Low Can You Go: Balancing Performance with Anonymity in Tor. In *Privacy Enhancing Technologies Symposium (PETS)*, 2013.

[35] J. Geddes, R. Jansen, and N. Hopper. IMUX: Managing Tor Connections from Two to Infinity, and Beyond. In *Workshop on Privacy in the Electronic Society (WPES)*, 2014.

[36] A. Ghedini. Encrypt it or Lose it: How Encrypted SNI Works. https://blog.cloudflare.com/encrypted-sni/, September 2018. Blog Post.

[37] D. Gopal and N. Heninger. Torchestra: Reducing Interactive Traffic Delays over Tor. In *Workshop on Privacy in the Electronic Society (WPES)*, 2012.

[38] D. Goulet. Ongoing DDoS on the Network. Tor-Project Email 001604, December 2017. https://lists.torproject.org/pipermail/tor-project/2017-December/001604.html.

[39] D. Goulet. Denial of Service mitigation subsystem. Tor Trac Ticket 24902, 2018. https://trac.torproject.org/projects/tor/ticket/24902.

[40] D. Goulet. Circuit cell queue can fill up memory. Tor Trac Ticket 25226, 2018. https://trac.torproject.org/projects/tor/ticket/25226.

[41] J. Hayes and G. Danezis. k-fingerprinting: a Robust Scalable Website Fingerprinting Technique. In *USENIX Security Symposium*, 2016.

[42] D. Herrmann, R. Wendolsky, and H. Federrath. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier. In *Workshop on Cloud Computing Security*, 2009.

[43] N. Hopper, E. Y. Vasserman, and E. Chan-Tin. How Much Anonymity Does Network Latency Leak? *ACM Transactions on Information and System Security (TISSEC)*, 13(2):13, 2010.

[44] C. Huitema and E. Rescorla. SNI Encryption in TLS Through Tunneling. Internet-Draft draft-ietf-tls-sni-encryption-02, Internet Engineering Task Force, 2018.

[45] R. Jansen. New Tor Denial of Service Attacks and Defenses. https://blog.torproject.org/new-tor-denial-service-attacks-and-defenses, January 2014. Blog Post.

[46] R. Jansen and R. Dingledine. Authenticating sendme cells to mitigate bandwidth attacks. Tor Proposal 289, 2016.

[47] R. Jansen and N. Hopper. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *Network and Distributed System Security Symposium (NDSS)*, 2012.

[48] R. Jansen, K. S. Bauer, N. Hopper, and R. Dingledine. Methodically Modeling the Tor Network. In *Workshop on Cyber Security Experimentation and Test (CSET)*, 2012.

[49] R. Jansen, P. Syverson, and N. Hopper. Throttling Tor Bandwidth Parasites. In *USENIX Security Symposium*, 2012.

[50] R. Jansen, J. Geddes, C. Wacek, M. Sherr, and P. Syverson. Never Been KIST: Tor's Congestion Management Blossoms with Kernel-Informed Socket Transport. In *USENIX Security Symposium*, 2014.

[51] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann. The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network. In *Network and Distributed System Security Symposium (NDSS)*, 2014.

[52] R. Jansen, M. Traudt, J. Geddes, C. Wacek, M. Sherr, and P. Syverson. KIST: Kernel-Informed Socket Transport for Tor. *ACM Transactions on Privacy and Security (TOPS)*, 22(1):3:1–3:37, December 2018.

[53] R. Jansen, M. Traudt, and N. Hopper. Privacy-Preserving Dynamic Learning of Tor Network Traffic. In *Conference on Computer and Communications Security (CCS)*, 2018. See also `https://tmodel-ccs2018.github.io`.

[54] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. Users Get Routed: Traffic Correlation on Tor By Realistic Adversaries. In *Conference on Computer and Communications Security (CCS)*, 2013.

[55] A. Johnson, R. Jansen, N. Hopper, A. Segal, and P. Syverson. PeerFlow: Secure Load Balancing in Tor. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2017(2), April 2017.

[56] S. Li, H. Guo, and N. Hopper. Measuring Information Leakage in Website Fingerprinting Attacks and Defenses. In *Conference on Computer and Communications Security (CCS)*, 2018.

[57] Z. Ling, J. Luo, W. Yu, X. Fu, D. Xuan, and W. Jia. A New Cell Counter Based Attack Against Tor. In *Conference on Computer and Communications Security (CCS)*, 2009.

[58] Z. Ling, J. Luo, W. Yu, X. Fu, W. Jia, and W. Zhao. Protocol-Level Attacks against Tor. *Computer Networks*, 57(4):869–886, 2013.

[59] A. Mani, T. W. Brown, R. Jansen, A. Johnson, and M. Sherr. Understanding Tor Usage with Privacy-Preserving Measurement. In *Internet Measurement Conference (IMC)*, 2018.

[60] B. Marczak, N. Weaver, J. Dalek, R. Ensafi, D. Fifield, S. McKune, A. Rey, J. Scott-Railton, R. Deibert, and V. Paxson. An analysis of China's "Great Cannon". In *Workshop of Free and Open Communication on the Internet (FOCI)*, 2015. See also `https://citizenlab.ca/2015/04/chinas-great-cannon`.

[61] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM Computer Communication Review*, 27(3):67–82, 1997.

[62] S. Matic, C. Troncoso, and J. Caballero. Dissecting Tor Bridges: a Security Evaluation of Their Private and Public Infrastructures. In *Network and Distributed System Security Symposium (NDSS)*, 2017.

[63] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov. Stealthy Traffic Analysis of Low-Latency Anonymous Communication using Throughput Fingerprinting. In *Conference on Computer and Communications Security (CCS)*, 2011.

[64] S. J. Murdoch and G. Danezis. Low-Cost Traffic Analysis of Tor. In *Symposium on Security and Privacy (S&P)*, 2005.

[65] S. J. Murdoch and P. Zieliński. Sampled Traffic Analysis by Internet-Exchange-Level Adversaries. In *Workshop on Privacy Enhancing Technologies (PET)*, June 2007.

[66] M. Nasr, A. Bahramali, and A. Houmansadr. DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning. In *Conference on Computer and Communications Security (CCS)*, 2018.

[67] P. H. O'Neill. Tor's Ex-director: 'The Criminal Use of Tor has Become Overwhelming'. `https://www.cyberscoop.com/tor-dark-web-andrew-lewman-securedrop`, May 2017. Cyberscoop Online News Article.

[68] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. *ACM Computer Communication Review*, 28(4):303–314, 1998.

[69] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Workshop on Privacy in the Electronic Society (WPES)*, 2011.

[70] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel. Website Fingerprinting at Internet Scale. In *Network and Distributed System Security Symposium (NDSS)*, 2016.

[71] V. Pappas, E. Athanasopoulos, S. Ioannidis, and E. P. Markatos. Compromising Anonymity using Packet Spinning. In *International Conference on Information Security*, 2008.

[72] M. Perry. TorFlow: Tor Network Analysis. In *Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs)*, 2009.

[73] K. Poulsen. Feds Are Suspects in New Malware That Attacks Tor Anonymity. `https://www.wired.com/2013/08/freedom-hosting`, August 2013. Wired Online News Article.

[74] J. J. Santanna, R. van Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Z. Granville, and A. Pras. Booters—An analysis of DDoS-as-a-service attacks. In *Integrated Network Management (IM)*, 2015.

[75] J. J. Santanna, R. d. O. Schmidt, D. Tuncer, A. Sperotto, L. Z. Granville, and A. Pras. Quiet Dogs Can Bite: Which Booters Should We Go After, and What Are Our Mitigation Options? *IEEE Communications Magazine*, 55(7):50–56, 2017.

[76] B. Schneier. Attacking Tor: how the NSA targets users' online anonymity. `https://www.theguardian.com/world/2013/oct/04/tor-attacks-nsa-users-online-anonymity`, October 2013. The Guardian Online News Article.

[77] P. Sirinam, M. Imani, M. Juarez, and M. Wright. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. In *Conference on Computer and Communications Security (CCS)*, 2018.

[78] R. Snader and N. Borisov. EigenSpeed: Secure Peer-to-Peer Bandwidth Evaluation. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2009.

[79] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal. RAPTOR: Routing Attacks on Privacy in Tor. In *USENIX Security Symposium*, 2015.

[80] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an Analysis of Onion Routing Security. In *Designing Privacy Enhancing Technologies*, 2001.

[81] H. Tan, M. Sherr, and W. Zhou. Data-plane Defenses against Routing Attacks on Tor. *Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2016(4), 2016.

[82] C. Wacek, H. Tan, K. Bauer, and M. Sherr. An Empirical Evaluation of Relay Selection in Tor. In *Network and Distributed System Security Symposium (NDSS)*, 2013.

[83] R. Wails, Y. Sun, A. Johnson, M. Chiang, and P. Mittal. Tempest: Temporal Dynamics in Anonymity Systems. *Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2018(3), 2018.

[84] T. Wang and I. Goldberg. Improved Website Fingerprinting on Tor. In *Workshop on Privacy in the Electronic Society (WPES)*, 2013.

[85] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. Effective Attacks and Provable Defenses for Website Fingerprinting. In *USENIX Security Symposium*, 2014.