Aaron Johnson*, Rob Jansen, Nicholas Hopper, Aaron Segal, and Paul Syverson

PeerFlow: Secure Load Balancing in Tor

Abstract: We present PeerFlow, a system to securely load balance client traffic in Tor. Security in Tor requires that no adversary handle too much traffic. However, Tor relays are run by volunteers who cannot be trusted to report the relay bandwidths, which Tor clients use for load balancing. We show that existing methods to determine the bandwidths of Tor relays allow an adversary with little bandwidth to attack large amounts of client traffic. These methods include Tor's current bandwidth-scanning system, TorFlow, and the peer-measurement system EigenSpeed. We present an improved design called PeerFlow that uses a peer-measurement process both to limit an adversary's ability to increase his measured bandwidth and to improve accuracy. We show our system to be secure, fast, and efficient. We implement PeerFlow in Tor and demonstrate its speed and accuracy in large-scale network simulations.

Keywords: Tor, distributed systems, security

DOI Editor to enter DOI
Received ..; revised ..; accepted ...

1 Introduction

Tor [12] is a popular anonymous-communication network. It consists of over 7000 volunteer relays carrying the traffic of over 2 million users daily at over 60Gib/s on average [1]. To balance this large traffic load over a diverse relay population, Tor estimates relay bandwidth using both self measurements and external measurements and then directs clients to use relays in proportion to the relays' bandwidth estimates. This load-balancing system is an attractive target for attack because the relays that carry a client's connection are able to learn sensitive properties about that connection, such as the client's identity. If an adversary controls enough of those relays, he can *deanonymize* the connection [24].

Tor designed its current relay-measurement system, TorFlow [5, 26], to avoid relying entirely on self-reported measurements [7] and thereby improve performance and security. TorFlow implements "bandwidth scanning", in which measurement authorities create connections through relays to measure their bandwidth. Researchers have observed [8, 31] that in this system a malicious relay can increase its apparent bandwidth. We confirm this observation by implementing and experimentally testing the attacks. Our results show that an adversary can obtain 177 times more client traffic than he should. An adversary with 1% of the network bandwidth could have 64% of client connections directed to him, giving him the opportunity to attack a large majority of Tor users. Moreover, the adversary can effectively attack this large number of connections even with his small amount of bandwidth, for example attempting to deanonymize connections using a correlation attack that completes before application data is sent [7] and then immediately shutting the TCP window.

The main alternative to TorFlow is the EigenSpeed system of Snader and Borisov [27–29]. In EigenSpeed, each Tor relay measures the speeds of its connections to other Tor relays and reports them to an authority, who applies Principal Component Analysis (PCA) to produce bandwidth estimates. We show that EigenSpeed too is highly vulnerable to attack. We identify basic flaws in its measurement method and initialization process, and we describe and experimentally demonstrate fundamental flaws in using PCA to aggregate measurements. These flaws allow an adversary to either get most honest relays kicked out of the network or receive up to 420 times more client traffic than he should.

We present PeerFlow, a load-balancing scheme for Tor that prevents an adversary from being directed a share of client traffic that is much larger than his share of the network capacity. PeerFlow uses a bandwidth-weighted voting process that resists manipulation by low-bandwidth adversaries, and it can use measurements from trusted relays for improved security. PeerFlow solves many of the additional challenges to creating a complete replacement for TorFlow, including a secure process to bootstrap new relays and techniques to ensure user privacy in reported traffic statistics. We also prototype PeerFlow in the actual Tor software and

***Corresponding Author: Aaron Johnson:** U.S. Naval Research Laboratory, E-mail: aaron.m.johnson@nrl.navy.mil
Rob Jansen: U.S. Naval Research Laboratory, E-mail: rob.g.jansen@nrl.navy.mil
Nicholas Hopper: University of Minnesota, E-mail: hopper@cs.umn.edu
Aaron Segal: Yale University, E-mail: aaron.segal@yale.edu
Paul Syverson: U.S. Naval Research Laboratory, E-mail: paul.syverson@nrl.navy.mil

use large-scale network simulations to show that its load balancing maintains Tor's current performance.

2 Background and Related Work

Tor: Tor [12] anonymizes a client TCP connection by randomly choosing three *relays* from its network, creating a multiply-encrypted *circuit* over those relays, and creating a *stream* through that circuit, which causes the endpoint to create an associated TCP connection to the destination. Streams can be multiplexed over a circuit.

Clients have a small, longstanding set of relays (*guards*) from which they select the first hop on their circuits. Tor clients use one guard rotated every 2–3 months. To become guards, relays must be old enough, provide at least 250KB/s of bandwidth, and have adequate uptime. Relays meeting these criteria receive the Guard flag, which makes them eligible to be selected as guards. Circuits also have a second (*middle*) hop and a third (*exit*) hop. Relays must allow connection to the client's desired destination port and IP to be selected as the exit. Most exits receive the Exit flag, given when a relay allows exit to the most useful ports. Currently relays must also receive the Fast flag (which requires a bandwidth of 100KB/s) to be selected at all.

A system of *Directory Authorities* maintains flag and other relay information and publishes an hourly *consensus*, which clients use to select relays for circuits. Clients choose relays for a circuit position randomly with probability roughly proportional to the product of each relay's *consensus weight*, which is proportional to the relay's bandwidth and is for load balancing in a given position, and a *position weight*, which is based on the relay's flags and is used to improve load balancing across the different relay positions (namely, guard, middle, and exit). Relays also provide the Directory Authorities with a self-determined *advertised bandwidth* to aid in setting their consensus weights.

Tor security requires that most of the network by consensus weight is not malicious. An adversary that controls much of the network will be selected often by clients. When selected as a guard, he can apply website fingerprinting [32, 33] to identify the client's destination. When selected as guard and exit, he can deanonymize the connection using a first-last correlation attack [24]. And, of course, when selected for all three relays on a circuit, the connection can be trivially deanonymized.

Bandwidth Measurement: TorFlow [5, 26] is a bandwidth-scanning system currently used by Tor in which measuring authorities measure the bandwidths of Tor relays for the purpose of determining consensus weights and flags. Past work has shown that TorFlow is vulnerable to multiple attacks [7, 31], which we will explore in more detail in §3. In contrast, PeerFlow uses relays' observations of each other in order to determine relay bandwidths. EigenSpeed [27–29] is a proposed scheme for secure bandwidth estimation that — like PeerFlow — uses peer measurements. Presented as a scheme for use in the Tor network, it is also proposed more generally for use in peer-to-peer distribution networks. EigenSpeed is the state of the art for this problem as far as we are aware, and it has been considered for adoption into Tor.[1] We show in §4 that EigenSpeed can be manipulated through several attacks and does not achieve its security goals.

Other: Karame et al. [25] describe attacks on link capacity estimation techniques and suggest using trusted network hardware to secure these measurements. Suselbeck et al. [30] propose a system for estimating peer bandwidth in a P2P system. It includes both passive measurements and active traffic injection, as PeerFlow does, but it assumes all peers are trustworthy, which PeerFlow does not. Haeberlen et al. propose the PeerReview system [17], which users cryptographic logs of node activity and witness audits of a node's actions to detect misbehavior in a distributed system. While these methods might further enhance the security of PeerFlow, they are unable to solve major challenges, including (*i*) exposing falsely-claimed transfers between malicious relays and (*ii*) identifying trustworthy witnesses in a system where Sybil attacks are possible. Jansen et al. [22, 23] describe how secure bandwidth measurements in Tor could be used to build a system to incentivize Tor relay operators by rewarding them for transferring traffic.

3 Attacks on TorFlow

TorFlow: TorFlow [5, 26] is a tool used by bandwidth-measuring Directory Authorities to directly measure the bandwidths of Tor relays. TorFlow works by constructing a series of *measurement circuits*, using them to perform test downloads, and then computing a weight for each relay based on the speeds of the test downloads.

TorFlow selects which relays to measure by dividing the list of all relays into *slices* of 50 relays of similar bandwidth (according to the most recent consensus). It measures each slice by constructing two-hop measurement circuits using only relays from that slice. When

[1] https://trac.torproject.org/projects/tor/ticket/5464

each circuit is built, TorFlow uses it to download a file from `torproject.org`. TorFlow continues building new circuits, choosing unmeasured relays from the current slice at random, to measure two relays at a time until every relay in the slice under examination has been measured several times. Then, for each relay, TorFlow takes the average bandwidth measured on circuits involving that relay, and stores this measurement to disk.

Every hour, TorFlow aggregates these measurements and produces a weight for each relay. A relay's weight is calculated by multiplying the relay's self-advertised bandwidth by the ratio between its measured bandwidth and the averaged measured bandwidth over the entire network. The Directory Authorities use these weights to produce the consensus weights.

Attacks: TorFlow allows a number of attacks that make it easy to manipulate a relay's apparent bandwidth. First, malicious relays can perform a *liar attack* [7], wherein they dishonestly report a higher bandwidth than they have available to increase their chances of being chosen during path selection while expending very few resources. TorFlow attempts to address this problem by adjusting self-reported bandwidths by a multiplier representing relative performance, but by continuing to use the reported bandwidth as a baseline, it remains vulnerable to the same attack. For example, a relay providing only 100 KB/s of bandwidth (the minimum required to obtain the Fast flag) could advertise a bandwidth so high that even after being adjusted down by TorFlow (as explained above), it will only be capped by 10 MB/s—the upper bound the Directory Authorities will assign to any relay. This attack is very effective, but gross exaggeration could be detected.

A more subtle attack takes advantage of TorFlow's two-hop measurement circuits, which are built with one of a small number of authorities on one end and a fixed URL on the other. Because the IP addresses of these measurement nodes are known, relays can easily recognize measurement circuits and treat them differently from ordinary ones, a technique demonstrated by Thill [31]. In a *selective denial-of-service (DoS) attack*, a relay can provide service only to measurement circuits while dropping all others, thereby giving the authorities the impression of excess capacity at very low cost. An adversarial exit can further reduce resource consumption by spoofing short responses from the destination instead of downloading and serving real ones, since TorFlow does not verify certificates or check the correctness or length of downloads.

Results: To demonstrate the efficacy of both the liar and selective DoS attacks, we developed a new TorFlow plug-in for the Shadow [3, 21] discrete-event network simulator. The plug-in mimics the functionality of the python scripts [5] that are used to run TorFlow in the public Tor network. Using Shadow and our new plug-in, we constructed a private Tor network with 498 relays, 4 directory authorities, 7500 clients, and 1 bandwidth authority that runs TorFlow. More details about our Tor model and simulations can be found in §7.

	Goodput (MiB/s)		Consensus Weight (%)	
	Median	Std. Dev.	Median	Std. Dev.
Baseline	22.5	5.9	7	1
Attack	0.2	0.1	11	5

Table 1. A relay can inflate its consensus weight at little cost by lying about its capacity and denying service to all but measurement circuits. Our experiments led to a bandwidth inflation factor of 177.

We implemented both attacks outlined above in Tor, and compiled the Shadow Tor plug-in with our modified Tor code. We arbitrarily chose one exit relay that was generated by Shadow's network generator to act as our test relay; this relay had an access link of 300 Mbits/s. We ran three baseline experiments in which our test relay acted completely honestly, and we ran six attack experiments in which our test relay ran both the liar and selective DoS attacks by: (i) only forwarding traffic for the TorFlow measurement circuits; and (ii) falsely reporting a bandwidth of 125000 KB to the directory authorities. We monitored the test relay's bandwidth usage and the fraction of the total consensus weight that our test relay achieved over time.

The results shown in Table 1 indicate that, using the attacks, the exit node was successfully able to *inflate* its consensus weight relative to other relays while at the same time consuming *significantly less* bandwidth (only what was required for the measurement circuits). Our attacks reduced the median bandwidth consumed by our test relay from 22.5 MiB/s to 0.2 MiB/s, while the median consensus weight obtained increased from 7% to 11%; our attack enabled the test relay to obtain more units of consensus weight fraction per bandwidth unit cost with a bandwidth inflation factor of 177. While we used only one relay for demonstration purposes, an adversary could use the same techniques with several relays to gain an even larger total fraction of the consensus weight.

We note that while TorFlow's current design makes these attacks easy to carry out, any bandwidth-scanning approach will need to solve the problem of relays detecting and favoring measurement probes, as observed by Biryukov *et al.* [8]. A major obstacle is the fact that some relays are unlikely to be chosen by normal clients in the middle circuit position (*e.g.* guards and exits), and measurement traffic must behave the same, exposing measurement sources or destinations to malicious relays and helping them identify the measurement probes.

4 Attacks on EigenSpeed

EigenSpeed: EigenSpeed uses as consensus weights the eigenvector of a matrix derived from a bandwidth matrix T, where T_{ij} is the bandwidth estimate of relay j by relay i. This process can be viewed as finding weights that are consistent with using themselves in a weighted average of the bandwidth estimates. Tor's Directory Authorities are the recipients of each node's estimates of the others and are responsible for determining the weights. Snader's thesis [27] is the final and most complete description of EigenSpeed, and so we use it as the as the authoritative version.

The bandwidth estimates are obtained from periodic measurements of the current bandwidth of an individual stream with a relay, rather than that relay's entire bandwidth (see [27] §3.3.1). Each new measurement is combined with a running bandwidth estimate using an exponentially-weighted moving average (EWMA). At the end of the measurement period, the estimates are put into matrix T, which the Directory Authorities use to produce $\overline{T}$ by first setting $\overline{T}_{ii} = 0$ to ignore self-measurements, next setting $\overline{T}_{ij} = \min(T_{ij}, T_{ji})$ to enforce symmetric measurements, and finally normalizing rows of $\overline{T}$ to sum to one. The EigenSpeed consensus weights are the left principal eigenvector v^* of $\overline{T}$, which is produced by iteratively computing $v^i = v^{i-1}\overline{T}$, where v^0 has a $1/t$ entry in the positions of a set of t *trusted relays* and a 0 entry elsewhere. Any relay j with measurements sufficiently different from the eigenvector, that is, with $\|v^* - \overline{T}_j\|_4 > L$ for $L = 10^{-5}$, is considered a liar and is removed and added to a set of *unevaluated* relays. Let $\|v^* - \overline{T}_j\|_4$ be the *liar metric* for j. Similarly, any relay j whose weight increased too fast during the first two iterations, that is, with $(v_j^2 - v_j^1)/v_j^1 > \Delta$ for $\Delta = 0.1$, is judged to be malicious and is removed and considered unevaluated. Let $(v_j^2 - v_j^1)/v_j^1$ be the *increase metric* for j. The unevaluated set also includes relays that are not in the largest component of the *measurement graph* (*e.g.* new relays), where an edge (i, j) exists in the graph if $\overline{T}_{ij} > 0$. In EigenSpeed, unevaluated relays will each get $1/n$ of the total consensus weight, where n is the total number of relays.

Attacks: An obvious vulnerability of EigenSpeed is that it selects each unevaluated relay with probability $1/n$. An adversary can flood the network with a large number of new relays contributing little or no bandwidth and thereby obtain a large total selection probability. Each new relay need only have a unique IP address. This could, for example, allow a botnet of just 20,000 computers to have a collective consensus weight of over 74% in a Tor network of its current size of about 7,000 honest relays. We therefore assume that unevaluated relays are selected with very low probability (*e.g.* if there are u unevaluated relays, then the probability of selecting a given one is $0.01/u$). This limits the effect of a Sybil attack to just taking over the unevaluated set, but it also means that a relay that is unevaluated can be essentially shut out of the Tor network.

We show how the two mechanisms that EigenSpeed uses to detect malicious nodes can be used to frame honest relays and have them put into the unevaluated set. The liar threshold $L = 10^{-5}$ and increase threshold $\Delta = 0.1$ are designed for an attack in which a clique of malicious relays report high bandwidth with each other and low bandwidth with others. However, modifications to this attack can confuse the trusted relays (which otherwise help distinguish between the honest and dishonest relays) by making the liar or increase metrics for honest framed relays appear large. This will imply that either (*i*) the adversary can get the honest non-trusted relays effectively kicked out of the network by moving them to the unevaluated set or (*ii*) L and Δ are large enough that the adversary can greatly inflate the inferred bandwidth of his relays.

Analysis: We demonstrate framing attacks using the Tor 2015-04-31 23:00 network consensus [2], which contains 5589 relays with positive selection probability (over 1000 relays have zero selection probability). EigenSpeed takes measurements of the per-stream bandwidth at a given relay rather than the total bandwidth at that relay. Thus we consider the ideal load-balanced case, in which all streams have the same bandwidth. We note that our attacks are even more effective when the network is not load balanced, in which case there is disagreement among the relays' observations, forcing the thresholds L and Δ to be large. For example, suppose that the relay observations are the total relay bandwidths, which we take to be the "observed bandwidths" in the relays' descriptors [4]. Then, with 10% of relays trusted (starting with the largest by observed bandwidth) and no malicious relays, honest relays have a liar metric as large as 0.00165, much larger than the suggested threshold of $L = 10^{-5}$, and honest relays have

# trusted relays (% of honest)	# adv relays	# framed relays	Adv bw %
280 (5%)	447	1118	1.92
559 (10%)	558	1118	2.83
1118 (20%)	558	559	2.83
1677 (30%)	558	112	3.00

Table 2. Cases with minimum bandwidth in which all framed relays had increase metrics above 0.2.

# trusted relays (% of honest)	# adv relays	False bw factor	Adv bw %	Adv weight %
280 (5%)	4191	100	3.49	98.2
559 (10%)	2235	100	3.70	93.9
1118 (20%)	1117	100	3.70	79.5
1677 (30%)	1397	15	6.52	48.5

Table 3. Cases with maximum weight in which all malicious relays had increase metrics below 0.2 and liar metrics below the honest non-trusted relays.

an increase metric as large as 0.86, much larger than the suggested $\Delta = 0.1$. We thus consider all our attacks in the load-balanced setting, in which case these metrics are much smaller and well below the suggested thresholds. In all of our experiments, we follow Snader [27] and stop the iterative eigenvector calculation when the change in vector norm is less than 10^{-10}.

The *increase framing attack* causes the increase metric of honest relays in a targeted set to be large. In this attack, a set of malicious relays uses sufficient bandwidth to obtain the average true bandwidth of honest relays with all trusted relays and with a subset of "framed" honest non-trusted relays. The malicious relays also falsely claim that bandwidth measurement among themselves (*i.e.* no data is actually sent among them). All other measurements with malicious relays are zero. All measurements among the honest relays are the same load-balanced flow rate (say, 1). Note that each malicious relay can easily obtain any bandwidth measurement with any honest relay, as long as the measurement does not exceed the relay's true bandwidth, by creating spurious connections to the honest relay with the necessary amounts of traffic. The malicious relay may furthermore drop all connections from honest clients for measurements of zero.

We consider adding relays to the Tor network in order to frame a subset of honest relays by causing their increase metric to reach above 0.2. Not only is this amount is greater than the $\Delta = 0.1$ recommended, it will be enough to allow significant bandwidth inflation and thereby demonstrate that no setting for Δ can provide good protection. For various numbers of trusted relays, we searched for a minimum adversarial BW needed to frame at least 2% of the relays. We count the bandwidth of a relay as the sum of the estimates obtained with each other relay, ignoring the false estimates among malicious relays. Table 2 presents our results. It shows that with at most 3% of the total bandwidth, the adversary can frame between 112 (2%) and 1118 (20%) of the honest relays, with the number decreasing as the number of honest relays that are trusted increases from 280 (5%) to 1677 (30%). The number of relays that the adversary must add is not large, from 447 to 558. Moreover, the attack can easily be repeated (with a different set of malicious relays) in order to move even more honest relays into the unevaluated set. As we noted, the selection probability for relays in the unevaluated set must be quite low. Therefore, they will be rarely observed by other relays, and so they must either wait many measurements periods to be evaluated or will have very low inferred bandwidths. In addition, if the adversary performs another Sybil attack on the unevaluated pool, which requires IP addresses but no bandwidth, then relays in the unevaluated pool are effectively removed entirely. Even worse, the relay bandwidths are highly skewed, and so even for the smallest number of framed relays in Table 2 (112), the adversary can quickly cause most of the network capacity to be unused. For example, in the consensus used in our experiments, 50% of the total observed bandwidth is provided by the largest 464 relays and 75% by the largest 1172.

We next show the *targeted liar attack*, in which just the trusted relays are provided a high-bandwidth flow. This targeting of trusted relays keeps the liar metrics of the malicious relays low while inflating the malicious relays' weight. In these experiments, the malicious relays obtain bandwidth estimates with the trusted relays that are the average bandwidth of the honest non-trusted relays, the malicious relays create measurements of zero with the honest non-trusted relays, and the malicious relays report a large false bandwidth among themselves. All honest relays again have the same bandwidth measurement among themselves.

Thus the trusted relays make the same measurements with the other honest relays as with the malicious relays, but the measurements of the honest non-trusted relays and those of the malicious relays disagree. This makes the attack use less bandwidth while keeping the liar metrics of malicious relays close to those of honest non-trusted relays. Table 3 lists the scenarios in which the final adversary weight was maximized among those we tested subject to (i) preferring adversary bandwidth of less than 4% when possible, (ii) producing increase metrics for malicious relays of no larger than 0.2, and (*iii*) producing liar metrics of malicious relays less than those of honest non-trusted relays. We can assume that $\Delta > 0.2$, or the previously-described increase framing attack would be possible. Therefore, no malicious relays are unevaluated due to an increase metric above Δ. Furthermore, no setting of the liar threshold L is able discriminate between malicious and honest relays. Either it would allow the large adversary weight inflations shown in Table 3 or it would put all honest non-trusted relays in the unevaluated set. We can see that by reporting a false bandwidth of 15-100 times the load-balanced rate reported by all honest relays, malicious relays can obtain weight inflated 7.4-28.1 times.

5 PeerFlow

PeerFlow uses two relay-measurement techniques: (*i*) each relay reports on the number of bytes it sent to or received from other relays, and (*ii*) each relay reports its available but unused bandwidth. Measurements from the first technique are combined to estimate the total bytes transferred after trimming a weight fraction λ of the largest and smallest values so that an adversary without λ of the network capacity cannot manipulate the outcome. Measurements from trusted relays (if available) are used to ensure that the estimates of bytes transferred are not unreasonably high or low. The second measurement technique allows the network to discover any unused bandwidth. It is vulnerable to lying by an adversary, though, and therefore PeerFlow will only consider increasing the consensus weight for a relay after consulting the secure outputs of the first measurement technique and verifying that the relay carried the expected amount of traffic. Further methods are used to increase the privacy, accuracy, and speed of these measurements and to securely bootstrap new relays into the system. We now explain each of the PeerFlow components in detail. For convenience, the key variables and parameters are listed in Table 4.

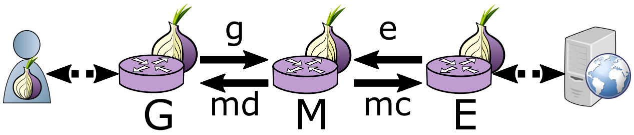


Fig. 1. Measuring positions in a circuit: relay G in position g measures relay M, relay M in position *mc* measures relay E, relay M in position *md* measures relay G, and relay E in position e measures relay M. A dashed arrow indicates hosts between which traffic is not measured.

Name	Description
β_p^{SR}	*bytes* measured by S to and from R in position p
$\rho_{(p)}^{(S)R}$	total traffic *relayed* by R (inferred by S, in position p)
$\eta^{(S)R}$	*estimated* traffic relayed by R (with S)
κ^R	*capacity* of relay R computed by Directory Authorities
σ^R	*self*-estimated capacity of relay R
ω^R	consensus *weight* of router R
v_p^R	*voting* weight of relay R in position p
t^R	measurement *time* of relay R
λ	fraction of measuring relay inferences to trim
τ	trusted relay weight fraction in each position
ϵ_{dec}	max fraction a relay's weight can decrease
μ	weight of measuring relays: 0.75
ϵ_{inc}	max fraction a relay's weight can increase: 0.25
δ_{noise}	traffic amount covered by differential privacy: 1 MiB
ϵ_{noise}	differential privacy guarantee: 0.1
ϵ_{err}	targeted fraction of traffic added as noise: 0.1
q_{err}	probability noise greater than targeted fraction: 0.1
ϵ_{loss}	accuracy loss from maximum of noisy values: 0.01

Table 4. Key variables (top), parameters (bottom) in PeerFlow

5.1 Measuring total traffic of a relay

For each circuit position, a subset of relays keeps track of the amount of traffic it sends to and receives from all of the other public relays while in that position. Let $\mathcal{M}_g$, $\mathcal{M}_m$, and $\mathcal{M}_e$ be the set of *measuring guards*, *measuring middles*, and *measuring exits*, respectively. $\mathcal{M}_p$ is defined to be the set containing each relay R with positive *voting weight* v_p^R in position p, that is, $\mathcal{M}_p = \{R | v_p^R > 0\}$. Voting weights approximate the relays' relative capacity in each position, with only the largest μ fraction of relays by capacity assigned non-zero voting weights to speed up measurement (see §5.6).

Each measuring relay $r \in \mathcal{M}_p$ counts the number of bytes exchanged with each other relay while r is in position p. Bytes are counted at the application layer, and both sent and received bytes are included in the count. A measuring relay detects itself in the guard position if the circuit-creation messages are not sent by a public Tor relay, it detects itself in the exit position if the circuit is not extended past the measuring relay, and otherwise it assumes the middle position. A measuring middle further divides its observations into *client-side* and *destination-side* measurements according to whether the measuring relay extended the circuit to the measured relay during circuit creation or vice versa, respectively. A measuring relay R_0 counts traffic sent to and received from each relay R_1 for a measurement period of length t_{R_1} that depends on the bandwidth of R_1 (see §5.4). As in TorFlow, the measurement periods generally span multiple consensuses and need not be aligned with them. Let $\beta_p^{R_0 R_1}$ be the number of bytes that measuring relay

R_0 sent to or received from R_1 during the measurement period while R_0 was in *directional position p*. The possible directional positions are guard, client-side middle, destination-side middle, and exit, denoted g, mc, md, and e, respectively (see Figure 1).

At the end of the measurement period for relay R_1, measuring relay R_0 will process the traffic count for each position that it measures for and send it to the Directory Authorities. The first step in processing count $\beta_p^{R_0R_1}$ protects the privacy of individual traffic flows by adding noise. The noise is a random value selected from the Laplace distribution $\mathsf{Lap}(b)$, which has mean 0 and variance $2b^2$, where b is set to provide differential privacy for a certain amount of traffic on the link (see §5.3). Let $N_p^{R_0R_1} \sim \mathsf{Lap}(b)$ be the noise value, and let $\tilde{\beta}_p^{R_0R_1} = \beta_p^{R_0R_1} + N_p^{R_0R_1}$.

The second processing step is to infer the total amount of traffic to or from R_1 seen in directional position p. This is accomplished by adjusting the amount R_0 sees by the probability of making that observation. Let $q_g^{R_0}$ be the probability of selecting R_0 as a guard, as determined from the consensuses of the measurement period. Let $q_m^{R_0}$, $q_{mc}^{R_0}$, and $q_{md}^{R_0}$ all be set to the probability of selecting R_0 as a middle. Let $q_e^{R_0}$ be the probability that R_0 is chosen as an exit (using the presence of the Exit flag as an approximate way to identify possible exits). Then let the estimate for the total traffic relayed by R_1 and seen in directional position $p \in \{g, mc, md, e\}$ be $\rho_p^{R_0R_1} = \tilde{\beta}_p^{R_0R_1}/q_p^{R_0}$.

At the end of the measurement period for R_1, the Directory Authorities will receive the ρ statistics about R_1 from the measuring relays. The Directory Authorities remove the largest and smallest ρ values for each directional position and aggregate the remaining values to obtain an estimate $\bar{\rho}_p^{R_1}$. When removing the extreme values, a voting weight v_p^R that is proportional to R's capacity is attached to each $\rho_p^{RR_1}$, and then a fraction λ of the voting weight is trimmed from both the top and bottom of the sorted ρ statistics. These voting weights help ensure that a low-capacity adversary has little effect on the measurement outcomes. To be more precise, let i_j be the index of the relay R_{i_j} with the jth largest value $\rho_p^{R_{i_j}R_1}$, let j_1 be the largest value such that $\sum_{j<j_1} v_p^{R_{i_j}} < \lambda$, and let j_2 be the smallest value such that $\sum_{j>j_2} v_p^{R_{i_j}} < \lambda$. The trimmed ρ statistics are aggregated by adding their noisy byte values and dividing by the total selection probability of the untrimmed relays: $\bar{\rho}_p^{R_1} = \sum_{j_1 \le j \le j_2} \rho_p^{R_{i_j}R_1} q_p^{R_{i_j}} / \sum_{j_1 \le j \le j_2} q_p^{R_{i_j}}$. Observe that j_1 and j_2 are defined such that, for all $\lambda \le 0.5$, some ρ value is untrimmed (*i.e.* $j_1 \le j_2$).

When there are no trusted relays, we set $\lambda = 0.256$ to maximize the size of the adversary that is prevented from arbitrarily increasing his weight (see §6). Note that using the median (*i.e.* $\lambda = 0.5$) does not provide optimal security in this case. When some positive fraction of the network τ is trusted in each position, we set λ to maximize the size of the adversary such that using μ of the network for measurement results in a smaller bound on an adversary's capacity increase compared to simply using measurements from the smaller trusted fraction τ. For $\tau = 0.05$, 0.1, 0.2, and 0.3, the respective values of λ are 0.34, 0.348, 0.497, and 0.498.

Given the aggregate values $\bar{\rho}_p^{R_1}$, the Directory Authorities calculate two estimates for the total number of bytes relayed by R_1. The first is the sum of the client-side estimates, $\rho_c^{R_1} = \bar{\rho}_g^{R_1} + \bar{\rho}_{mc}^{R_1}$, and the second is the sum of the destination-side estimates, $\rho_d^{R_1} = \bar{\rho}_{md}^{R_1} + \bar{\rho}_e^{R_1}$. If R_1 can act as a guard, then client-side observations for it will be missing for any circuits on which it is a guard, and similarly for destination-side observations if R_1 can act as an exit. On the other hand, when a relay acts as a middle it is observed both on the client and destination side but should get credit for that traffic only once. Therefore, the Directory Authorities use $\rho_{\max}^{R_1} = \max(\rho_c^{R_1}, \rho_d^{R_1})$ as an estimate for the total amount of traffic relayed by R_1. To avoid excluding client-side or destination-side observations made on separate circuits, PeerFlow requires that during a measurement period a Tor relay will not operate both as a guard and as an exit (Tor effectively already enforces this currently via its bandwidth weights, as exit bandwidth is relatively scarce and is thus reserved for exiting).

PeerFlow takes advantage of measurements from any trusted relays by using them to limit the range in which a relay's traffic will be inferred. PeerFlow uses trusted relays in this way instead of simply using only the trusted measurements because doing so provides better security and accuracy when relatively little of the network is trusted. Because many of the highest-bandwidth Tor relays are managed by organizations closely aligned with the Tor Project, and (as of 15 May 2015) the top 60 relays constitute over 20% of the total weight, it seems reasonable to imagine a set of trusted relays that carry 15-25% of Tor traffic.

Let $\mathcal{T}_p$ be the set of trusted relays in position p, and let τ be the minimum fraction of relay capacity that they are assumed to provide in each of the guard, middle, and exit positions. At the end of a measurement period for R_1, the Directory Authorities simply combine the measurements from the trusted relays of bytes exchanged with R_1 to determine the following trusted estimate for the total bytes relayed by R_1 and observed in position p: $\hat{\rho}_p^{R_1} = \sum_{R \in \mathcal{T}_p} \rho_p^{RR_1} q_p^R / \sum_{R \in \mathcal{T}_p} q_p^R$. PeerFlow then combines these positional trusted estimates as it does with the analogous estimates from all relay measurements to produce $\hat{\rho}_c^{R_1} = \hat{\rho}_g^{R_1} + \hat{\rho}_{mc}^{R_1}$, $\hat{\rho}_d^{R_1} = \hat{\rho}_{md}^{R_1} + \hat{\rho}_e^{R_1}$, and $\hat{\rho}_{\max}^{R_1} = \max(\hat{\rho}_c^{R_1}, \hat{\rho}_d^{R_1})$.

The trusted estimate $\hat{\rho}_{\max}^{R_1}$ is used to adjust the all-relay estimate $\rho_{\max}^{R_1}$ by enforcing a ceiling and floor on its value. The ceiling is simply $\hat{\rho}_{\text{ceil}}^{R_1} = \hat{\rho}_{\max}^{R_1}$, which means that the inferred traffic relayed by R_1 will be limited by the number of bytes it can exchange with trusted relays. The floor is $\hat{\rho}_{\text{floor}}^{R_1} = \hat{\rho}_{\max}^{R_1} \tau / (\mu(1 - \lambda))$, which both limits the amount that the adversary can use falsely low measurements from its relays to reduce honest relays' inferred traffic relayed and ensures that the adversary gains no advantage in targeting its bandwidth on the trusted relays instead of the top $1 - \lambda$ fraction of measuring relays in a given position. The final estimate for the traffic relayed by R_1 is thus $\rho^{R_1} = \max(\min(\hat{\rho}_{\text{ceil}}^{R_1}, \rho_{\max}^{R_1}), \hat{\rho}_{\text{floor}}^{R_1})$. If there are no trusted relays, then $\rho^{R_1} = \rho_{\max}^{R_1}$.

5.2 Measuring available bandwidth

Each relay R monitors its network activity during a measurement period and estimates its total capacity for relaying traffic. That is, R estimates the maximum rate at which it could have relayed traffic during the measurement period if Tor clients had asked it to. This could be measured in the same way that Tor relays currently determine their self-advertised bandwidths. R sends this self-measured value σ^R to the Directory Authorities at the end of the measurement period.

5.3 Preserving link privacy with noise

The traffic statistics that measuring relays report to Directory Authorities reveal how traffic flows through the Tor network. Traffic statistics *per relay* are already collected and reported by Tor [2], but the PeerFlow statistics describe traffic between each *pair* of relays. These statistics should not assumed to be kept secret by the Directory Authorities, because some Directory Authorities may be compromised and because it allows auditing of PeerFlow to identify errors and relay misbehavior.

The risk of releasing the PeerFlow measurements is that they could be used to identify the routes through the Tor network taken by a target set of connections. For example, a malicious destination might target an incoming connection and use its observation of the exit node and traffic volume to identify as the middle node that relay measured by PeerFlow to have sent the same amount of traffic as the target connection. The guard relay could then be identified similarly. Although there exist other ways of identifying the relays used on a connection (*e.g.* the congestion attack [14], latency attack [18], and predecessor attack [34]), this information should still have some protection.

Therefore, measuring relays in PeerFlow add a random value to the observed byte totals. The goal of this added noise is to limit the certainty with which an adversary can conclude that a given client stream was carried between a given pair of relays. We accomplish this by choosing the noise value randomly from the Laplace distribution with mean 0, which has the probability density function $\mathsf{Lap}(b; x) = e^{-|x|/b}/(2b)$. Adding noise according to the Laplace distribution provides *differential privacy* [13], where the privacy notion applies to a given amount of traffic. We set the Laplace parameter to $b = \delta_{\text{noise}}/\epsilon_{\text{noise}}$, where δ_{noise} is the maximum amount of traffic for which ϵ_{noise}-differential privacy is provided (we use $\delta_{\text{noise}} = 1$ MiB and $\epsilon_{\text{noise}} = 0.1$). Providing differential privacy to Tor traffic statistics follows the suggestion of Goulet *et al.* [16]. Appendix A describes a cryptographic measurement-aggregation scheme that further limits the amount of traffic data revealed.

5.4 Measurement periods

The length of a measurement period is determined for each relay individually and is updated at the end of the each measurement period. We would like this length to be low in order to enable quick response to changes in relay bandwidth and load. The speed of measurement is limited by how quickly a relay can exchange an amount of client traffic with each measuring relay such that the added noise is relatively small.

Let t^{R_1} be the length of the next measurement period for relay R_1. The Directory Authorities determine t^{R_1} after inferring ρ^{R_1} by using ρ^{R_1} and the consensus weights to estimate the traffic rates with each measuring relay R_0 in each position p. t^{R_1} is set large enough such that with probability $1 - q_{\text{err}}$ for at least $1 - 2\lambda$ of the measuring relays by voting weight the amount of noise added is less than a fraction ϵ_{err} of the traffic exchanged with R_1 during the measurement period (we use $q_{\text{err}} = 0.1$ and $\epsilon_{\text{err}} = 0.1$). $1 - 2\lambda$ of the voting weight ensures accurate estimates remain after trimming the largest and smallest fraction λ of the ρ statistics. In addition, t^{R_1} must be set large enough to limit to ϵ_{loss} the loss in accuracy due taking the maximum of noisy values $\hat{\rho}_c^{R_1}$ and $\hat{\rho}_d^{R_1}$ (see App. B for details; we use $\epsilon_{\text{loss}} = 0.01$).

5.5 Load balancing using measurements

The Directory Authorities use the aggregate peer-measurement ρ^R and self-measurement σ^R to produce the consensus weight ω^R for relay R. Tor clients select relay R for a given position in a new circuit with probability proportional to ω^R (approximately, see §2). Thus, in order to balance the load across the available relays, the Directory Authorities attempt to determine consensus weights that are proportional to the bandwidth of each relay. They must do this even as relay capacities and network traffic change, and they must do it in a way that is secure against manipulation. They will accomplish this by computing and comparing the amounts of traffic a relay was expected to transfer, did transfer, and claims it can transfer.

Let η^R be the amount of traffic that R is expected to have relayed in the measurement period that has just completed. To determine η^R, let $\mathcal{P}^R$ be the set of relays (including R) that could be chosen for the same positions as R in their most-recently completed measurement period based on the Guard, Exit, and Fast consensus flags, where for this purpose a relay is considered to possess each of these flags if they exist for the majority of consensuses during the measurement period. Then let $\omega_0^{R'}$ be the weight used by $R' \in \mathcal{P}^R$ during its most-recently completed measurement period, let $t_0^{R'}$ be the length of that period, and let $\rho_0^{R'}$ be the inferred number of bytes relayed during that period. Finally, set η^R to be the voting-weight median of the set $\{\omega_0^R t_0^R \rho_0^{R'}/(\omega_0^{R'} t_0^{R'})\}_{R' \in \mathcal{P}^R}$, which is the set of inferred bytes transferred adjusted for differences in consensus weight and measurement-period length.

Let κ^R be the current estimate for the *capacity* of R, that is, the amount of traffic that R is capable of relaying. κ^R is set initially during the bootstrapping process. Let S^R be the measurement state of R. After the bootstrapping period has finished, S^R will only take values NORMAL and PROBATION (its use during bootstrapping is described in §5.7).

For $S^R =$ NORMAL, if R is non-trusted, Figure 2 describes how the Directory Authorities update the consensus weight ω^R, the relay state S^R, and the capacity estimate κ^R. Observe that κ^R is never set to be larger than the relay's self-estimate σ^R and that κ^R is increased to the observed rate ρ^R/t^R if κ^R is less than it. κ^R will only be decreased if R transfers a certain fraction ϵ_{dec} less than both the expected amount η^R and R's estimated limit $\kappa^R t^R$. This fraction is set to at most $\epsilon_{\text{dec}} \leq 1 - \tau/(\mu(1-\lambda))$ to protect honest relays from being forced into PROBATION by an adversary. Setting ϵ_{dec} this way protects an honest R because R will send the expected amount to the trusted relays, and so $(1-\epsilon_{\text{dec}})\eta^R = \hat{\rho}^R_{\text{floor}} \leq \rho^R$. Without trusted relays, ϵ_{dec} should be set to a small value to allow some natural variation in traffic amounts without allowing too much underperformance without penalty (*e.g.* $\epsilon_{\text{dec}} = 0.25$). If ρ^R falls below the required threshold but R indicates with σ^R that it believes it has a higher capacity than the amount measured, then R maintains its capacity but enters the PROBATION state. When the relay stays in the normal state, the final weight produced ω^R is only increased either to a newly-demonstrated capacity κ^R or, if R believes it has additional unused capacity, to a fraction ϵ_{inc} over the old capacity (we use $\epsilon_{\text{inc}} = 0.25$).

```
$\kappa'^R = \min(\max(\rho^R/t^R, \kappa^R), \sigma^R)$
$\omega^R = \min(\max((1+\epsilon_{\text{inc}})\kappa^R, \kappa'^R), \sigma^R)$
$\kappa^R = \kappa'^R$
**if** $\rho^R < (1-\epsilon_{\text{dec}})\min(\kappa^R t^R, \eta^R) \wedge \rho^R/t^R < \sigma^R$ **then**
    $S^R =$PROBATION
    $\omega^R = \kappa^R$
**end if**
```

Fig. 2. Non-trusted relay weight algorithm if $S^R =$ NORMAL

```
$\kappa^R = \min(\rho^R/t^R, \sigma^R)$
$S^R =$NORMAL
$\omega^R = \min((1+\epsilon_{\text{inc}})\kappa^R, \sigma^R)$
```

Fig. 3. Non-trusted relay weight algorithm if $S^R =$ PROBATION

Probation allows a relay to avoid a weight decrease due to a random or malicious lack of client traffic. When non-trusted relay R enters the probation state, it attempts to prove over the next measurement period that it is capable of transferring at a rate that is the minimum of its current capacity κ^R and its self-assessed capacity σ^R. To do so, R monitors the number of bytes that it transfers to other relays, and it exchanges (identified) dummy traffic as needed to convince all measuring relays in $\mathcal{M}_p$ for some p that it has relayed $\min(\kappa^R, \sigma^R)t^R$ bytes total. When $R' \in \mathcal{M}_p$ receives β dummy traffic from R, it echoes it back and adds $\mu\beta$ to $\beta_p^{R'R}$. The μ factor is needed because dummy traffic is only sent to measuring relays, and, because dummy traffic is otherwise treated the same as other traffic, it provides no additional opportunity for the adversary to cheat the system. As shown in Figure 3, at the end of the measurement period the relay leaves probation status and is assigned the capacity that is measured ρ^R/t^R, unless that exceeds its updated self-assessment.

Trusted relays also update their weights as shown in Figures 2 and 3, with the additional requirement that trusted relays maintain τ fraction of the weight (*i.e.* selection probability) in each position. After the update of ω^R for *any* relay R, an inflation factor is computed that is multiplied by the trusted-relay weights when computing selection probabilities q_p^R for $R \in \mathcal{T}_p$. The inflation factor is taken to be minimum value greater than one such that $\sum_{R\in\mathcal{T}_p} q_p^R \geq \tau$ for $p \in \{g, m, e\}$.

5.6 Updating voting weights

For accuracy and security, the voting weight of a relay should reflect the amount of bandwidth it has provided in a given position. Giving relays with high contributed bandwidth high voting weights improves accuracy because those relays have the most observations about other relays' activity. It improves security because it requires the use of costly bandwidth both for an adversary to obtain large weights for its relays and to affect the weight of other relays. Thus PeerFlow bases voting weights on previous estimates for the amount of relayed traffic. However, voting weights are updated more slowly than consensus weights in order to increase the upfront bandwidth cost of obtaining influence over consensus weights. In addition, a fraction of the smallest relays is excluded from voting to speed up measurement.

We need to maintain that the measuring relays constitute a significant fraction of the network. However, we would also like to update voting weights infrequently to force an adversary to relay traffic for a while before increasing his voting weight. To satisfy the former concern, we select a fraction μ of the network capacity in each position to receive a positive voting weight (we use $\mu = 0.75$). To satisfy the latter, we only update the voting weights when the cumulative selection probability of any $1 - \lambda$ of the voting weight in some position falls below $(1-\lambda)\mu$ (*i.e.* when $\sum_{R\in\mathcal{S}} q_p^R < (1-\lambda)\mu$ for some relay set $\mathcal{S}$ with voting weight at least $1 - \lambda$ in position p). This will ensure that a small adversary (*i.e.* one with voting weight $\alpha < \lambda$) cannot inflate his weight by targeting the $1 - \lambda$ fraction of measuring relays that have had their selection probability reduced the most.

The Directory Authorities initiate any update of the voting weights after determining the next consensus weights. During the update, each Directory Authority gives a positive voting weight to the relays that have the largest capacity and constitute a fraction μ of the network capacity. Let κ_p^R be the current estimated capacity κ^R of relay R multiplied by the position weight for R and $p \in \{g, m, e\}$. Let $i_{j,p}$ be the index of the relay with the jth largest value κ_p^R. Let j_p^* be the number of relays needed to reach a fraction μ of $\sum_R \kappa_p^R$. Relay $R_{i_{j,p}}$, $j \leq j_p^*$, is assigned a voting weight for position p of $v_p^{R_{i_{j,p}}} = \kappa_p^{R_{i_{j,p}}} / \sum_{k\leq j_p^*} \kappa_p^{R_{i_{k,p}}}$. Relay $R_{i_{j,p}}$, $j > j_p^*$, is assigned a voting weight for position p of $v_p^{R_{i_{j,p}}} = 0$.

5.7 Bootstrapping new relays

PeerFlow boostraps new relays into the system using the following staged process:

Initialization: A new relay that Tor would currently just add to the next consensus has its measurement state S^R initialized to UNKNOWN.

Unknown: In any consensus period a set of *Bandwidth Authorities*, such as those currently used by TorFlow, estimate the capacity of a relay R with S^R =UNKNOWN by downloading a set of test files of increasing size through a one-hop circuit consisting of R. The test file is obtained from a Bandwidth Authority itself, and the Bandwidth Authority should have enough capacity to measure a reasonable lower-bound on capacity for the largest relays. Tor's current restrictions on one-hop circuits can be avoided by having the Bandwidth Authority act both as the client and as a spurious second hop. To reduce the ability of the adversary to slow down the testing process, Bandwidth Authorities test relays in the order that they joined, and they only allow the download to take as long as it would take a relay with the minimum amount of bandwidth needed for the Fast flag (currently 100KB/s [4]).

Regardless of whether all downloads finished, at the end of the tests for relay R, the Bandwidth Authority estimates a sustainable rate for the relay κ^R as the minimum of the observed rate (*i.e.* the test bytes transferred over the time needed to transfer them) and the initial self-measured capacity σ^R. The weight of R is initialized to $\omega^R = \kappa^R$. The measurement time t^R is set as described in §5.4, except the traffic amounts ρ_p^R in each measurement position p (which do not exist initially) are estimated by using those amounts $\rho_p^{R'}$ from the other relays R' with the same flags as R after their most-recently completed measurement period. To estimate ρ_p^R from these, each $\rho_p^{R'}$ is normalized to $\omega^R \rho_p^{R'} / (t^{R'} \omega^{R'})$, and ρ_p^R is set to the vote-weighted median of these values. Finally, the state of R is updated to S^R = ESTIMATED.

Note that this stage is for performance reasons only. The amount of data downloaded during the tests is not high and is not intended as a security barrier. In addition, the measured relay is aware that the requested downloads are measurement tests performed by Bandwidth Authorities.

Estimated: Relay R with S^R =ESTIMATED starts being selected only for the middle position when doing so will not cause the selection probability of estimated relays to exceed some small amount (*e.g.* 5%). The middle position cannot observe the client or destination directly, and so PeerFlow allows relays to occupy the middle position based only on an insecure capacity estimate. The limit on the probability of estimated relays limits the extent of observations they can make. Note that estimated relays are not considered when determining the measuring relays, which, among other things, prevents an adversary from triggering a voting-weight update by flooding new relays.

Measuring relays measure R just as they do for relays in the NORMAL state. After t^R time, an estimate ρ^R is produced for the amount relayed by R using the same trimmed vote as for relays in the NORMAL state. Then the capacity estimate is updated to $\kappa^R = \min(\rho^R/t^R, \sigma^R)$, and the consensus weight is set to $\omega^R = \kappa^R$. Finally, the relay's state is updated to $S^R = \text{NORMAL}$.

6 Security analysis

The main security goal of PeerFlow is to ensure that an adversary that relays a fraction ϕ of Tor's traffic in a given position only obtains a total relative consensus weight of $\gamma\phi$, where γ is a small advantage multiple. We first examine consensus weights in a given voting-weight period (voting-weight periods are described in §5.6). We show bounds on γ, which, in addition to the limits imposed by trusted relays, show that if the adversary is small (*i.e.*, has a maximum voting-weight fraction of $\alpha < \lambda$), the advantage is bounded even without any trusted relays. We then examine the consensus weights across voting-weight periods and show that PeerFlow provides bounded advantage there as well.

6.1 Weights in a single voting-weight period

Let α_p, $p \in \{g, m, e\}$, be the voting weight fraction that the adversary's relays cumulatively possess in position p, and let $\alpha = \max_p \alpha_p$. Let A be an adversarial relay. Let β^A be the total number of bytes sent or received by A during its measurement period. Let $n_{\text{msr}} = \max(|\mathcal{M}_g| + |\mathcal{M}_m|, |\mathcal{M}_e| + |\mathcal{M}_m|)$, and let $S_j = \sum_{1 \le i \le n_{\text{msr}}} N_i^j$, $j \in \{1, 2\}$, where each N_i^j is an independent random variable distributed in same way as the PeerFlow noise values (*i.e.* with distribution $\mathsf{Lap}(\delta_{\mathsf{noise}}/\epsilon_{\mathsf{noise}})$). We use the variables in Table 4 to refer to those values in the current measurement period, and we denote by x_0 the value of variable x in the last measurement period (*e.g.* κ_0^R indicates the capacity inferred for R for its previous measurement period). Note that the proofs of Lemmas 1 and 2 and of Theorem 1 appear in Appendix B.

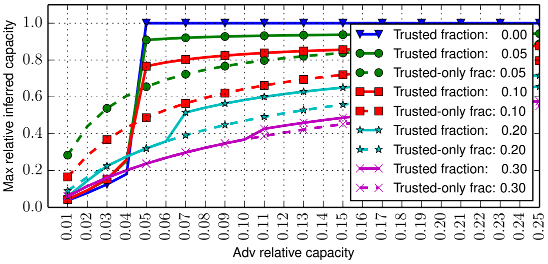


Fig. 4. Max adversarial relative inferred capacity across voting periods using trusted relays and using trusted relays *only*

We focus on $\alpha < \lambda$ because $\alpha = 0$ at the beginning of an attack, small adversaries will maintain small α (as shown in §6.2), and with $\alpha > \lambda$ the adversary's inferred capacity is bounded only by the trusted ceiling (*i.e.* the limit imposed by $\hat{\rho}^A_{\text{ceil}}$). Lemma 1 shows that the expected bytes that PeerFlow infers were transferred by A (*i.e.* ρ^A) are upper-bounded in some way by the number of bytes that A actually exchanged (*i.e.* β^A). As we will show, assuming A exchanges close to the amount necessary to stay out of probation, then the noise sums S_1 and S_2 will be small relative to β^A, and the upper bound in the lemma is roughly $\beta^A/((1 - \lambda - \alpha)\mu)$. For $\lambda = 0.256$, $\alpha < \lambda$, and $\mu = 0.75$, the adversary is thus limited to increasing ρ^A by a factor of roughly 2.73.

Lemma 1. *If $\alpha < \lambda$, then over the random noise values*

$$\mathbb{E}[\rho^A] \le ((1 - \lambda - \alpha)\mu)^{-1} \, \mathbb{E}\left[\max(S_1, \beta^A + S_2)\right].$$

Lemma 1 limits the amount an adversary can increase his relays' inferred bytes transferred. His total consensus weight is relative to the honest relays, however, and it may occur that many of these relays transfer most of their traffic with a small number of measuring relays, whose values get trimmed. However, it is unlikely that traffic from honest clients is skewed much in this way. Let $\gamma' \le 1$ be the smallest factor by which some set of honest measuring relays with voting weight $1 - 2\lambda$ underestimates an honest relay. Let $\mathcal{A}$ be the set of adversarial relays. Lemma 2 states that the inferred bytes of each honest relay is at most γ' of the true value $\beta^R/2$.

Lemma 2. *If $\alpha < \lambda$, then, for all $R \notin \mathcal{A}$, $\rho^R \ge \gamma'\beta^R/2$.*

These lemmas combine to prove Theorem 1, which states that either the malicious relay sends less than expected, putting it into probation, or it obtains a relative consensus weight at most a constant factor above the relative amount of traffic it transfers. With $\alpha = 0$,

$\gamma' = 1$, $\mu = 0.75$, $\epsilon_{\text{inc}} = 0.25$, $\lambda = 0.256$, and $\epsilon_{\text{loss}} = 0.01$, for example, the advantage factor is $\gamma = 4.52$.

Theorem 1. *For $\alpha < \lambda$ and over the random noise values, if $\beta^A < (1-\lambda-\alpha)(1-\epsilon_{dec})(1-\epsilon_{loss})\mu\rho_0^A t^A/t_0^A$, then $\mathbb{E}[\rho^A]$ puts A into probation, and otherwise*

$$\frac{\mathbb{E}[\omega^A]}{\sum_R \omega^R} \leq \left[\frac{2\gamma'(1+\epsilon_{inc})}{(1-\epsilon_{loss})(1-\lambda-\alpha)\mu}\right]\frac{(\beta^A/t^A)}{\sum_R(\beta^R/t^R)}.$$

While in theory the adversary can obtain the advantage factor γ in Theorem 1 (the value in brackets), doing so requires certain attacks that have costs and limitations. To obtain the factor of 2 in γ, the adversary must only send or receive "one-sided" traffic to measuring relays, *i.e.* only from the client side or destination side. In particular, he must not actually relay any Tor traffic. Acting this way is highly observable and is likely to be noticed by clients. To obtain the $((1-\lambda-\alpha)\mu)^{-1}$ factor in γ, the adversary must exchange traffic only with a set of measuring relays with voting weight $1-\lambda-\alpha$. This is observable by Directory Authorities (although hard to distinguish from a possible framing of an honest relay) and also noticeable by clients.

6.2 Weights across voting periods

We now consider the ability of an adversary to increase his inferred capacities κ^A (and thus his weights ω^A) over time, $A \in \mathcal{A}$. For this analysis, we ignore the possible inflation effect of the added noise (*i.e.* $\epsilon_{\text{loss}} = 0.01$), assume that honest relays send their share of the traffic volume to the measuring relays (*i.e.* that $\gamma' = 1$), and assume that voting weights and selection probabilities are maintained to be proportional (*e.g.* as when there is no network churn).

The adversary can apply a *feedback attack* to the bounds of Theorem 1 by repeatedly exchanging all traffic with a target set of relays with voting weight $1-\lambda-\alpha$, including the trusted relays, as α grows. He can accomplish this, for example, by severely rate limiting honest client connections and creating his own dummy connections through the targeted measuring relays. Initially, when $\alpha = 0$, this inflates his total capacity by a factor of at most $2/((1-\lambda)\mu)$. Then, once the adversary receives an inflated voting weight α, he can target $1-\lambda-\alpha$ of the measuring relays to exchange all traffic with, further increasing his voting weight by a factor of $(1-\lambda)/(1-\lambda-\alpha)$. The adversary can repeat this process until either he reaches a fixpoint or he receives voting weight $\alpha > \lambda$, at which point the capacities of his relays are either limited by the trusted-relay ceiling (*i.e.* $\hat{\rho}^A_{\text{ceil}}$) or are unbounded without trusted relays, and the adversary can start to deflate the weights of the honest relays, which are ultimately limited by the trusted-relay floor (*i.e.* $\hat{\rho}^R_{\text{floor}}$).

We simulate this process across voting periods until the adversary's weight no longer increases. We also consider a simpler variant of PeerFlow in which *only* trusted-relay measurements are used. In this variant, the trusted-relays measurements are used directly (*i.e.* $\rho^R = \hat{\rho}_{\max}$), there are no voting weights, and the other PeerFlow components operate in the same way.

The results are shown in Figure 4. It shows that PeerFlow can provide bounded weight inflation to small adversaries even when there are *no* trusted relays. Specifically, the inflation factor is $\gamma < 4.6$ when $\tau = 0$ and the adversary is at most 4% of the network. This compares favorably to the measured TorFlow and EigenSpeed inflation factors of up to 177 and 28.1, respectively. It also shows that using measurements from all relays protects against small adversaries at the expense of worse security against larger adversaries compared to using measurements from only trusted relays. Making this tradeoff is consistent with Tor's security in general [24]. We can also see that as the trusted fraction of the network grows, the security from using all relays and from using only trusted relays become the same. In both cases, as the trusted fraction grows, an adversary with ϕ of the network capacity has his inferred relative capacity limited to $(2\phi/\tau)/(1+2\phi/\tau)$, which implies a bounded inflation factor of $\gamma \leq 2/\tau$.

7 Load-Balancing Analysis

In this section, we experimentally demonstrate PeerFlow's ability to effectively distribute network traffic to relays in proportion to their bandwidth capacities.

7.1 Experimentation Setup

We evaluate PeerFlow and its effect on the consensus path selection weights, network goodput, and client performance using Shadow [21]. Shadow is an open-source [3] parallel discrete-event network simulator/emulator hybrid that has been extensively validated and utilized for Tor network experimentation [19, 20]. Shadow experiments are completely isolated from the network, so they are safe and contain no privacy risk to the public Tor network or its users. Because Shadow runs real applications as plug-ins, it is ideal for analyzing application-layer effects, *e.g.* those that result from modifying Tor's load-balancing protocols.

Our Private Tor Network: We generated a new Tor network configuration for Shadow using the tools in the Shadow distribution and Tor Metrics data [1] collected during 2014-09. The resulting Tor network contains 4 Tor directory authorities, 498 Tor relays, 7,500 Tor clients, and 1,000 servers. Our private network represents the public Tor network from 2014-09 at scale of approximately $\frac{1}{12}$, and we ran our private network for 5 virtual hours for each experiment (using the first two for network bootstrapping). We repeated each experiment 3 times to measure variability across runs, which we quantify for a given performance metric by comparing the distribution $F(x)$ across all 3 experiments to the distribution $F_i(x)$ for the ith experiment, $i \in \{1, 2, 3\}$, using the KS statistic: $\max_x |F(x) - F_i(x)|$.

We experiment with and compare three models to determine the effect of load-balancing weights on client performance and the distribution of network load. In the *Ideal* model, each relay's consensus weight is initialized to the capacity of its network interface as configured in Shadow, and these weights do not change throughout the experiment. In the *TorFlow* model, we run our new Shadow TorFlow plug-in that we developed[2] to mimic the behavior of TorFlow as it operates in the public Tor network. In the *PeerFlow* model, we run a PeerFlow prototype[3] that we implemented in Tor (forked at version `0.2.5.10`). We did not implement probation in our prototype. However, we do measure when probation would be triggered and how much traffic would be needed to avoid probation, and those measurements are presented in §8.2. In both the *TorFlow* and *PeerFlow* models, the bandwidth information used to produce the consensus weights is dynamically updated throughout the experiment. We run three experiments for each model using random seeds, and for each model we present the results aggregated across all three experiments. Note that we do not compare EigenSpeed because (i) it is not currently used in practice; and (ii) we believe that Tor will not adopt it at least partially due to the vulnerabilities discussed in §4.

7.2 Network Performance

A primary goal of Tor's network load-balancing algorithm is to distribute traffic to best utilize existing resources, and we use goodput as a metric for determining effective use of such resources. For our purposes, relay goodput, or application throughput, is the number of application bytes that a relay is able to forward over time. More formally, if a relay R_i receives $B_i^r(T)$ application bytes from the network and sends $B_i^s(T)$ application bytes to the network at time T over time interval t, then we define its goodput as $G_i(T) = min(B_i^r(T), B_i^s(T))/t$. We normalize t to 1 second for ease of exposition.

To determine how load is distributed over the relays, we compute each relay's per-second goodput $G_i(T)$ for every second of the final 3 virtual hours of simulation. Figure 5a shows the cumulative distribution of the goodput for each relay-second, over all relays, seconds, and experiments. The goodput achieved under all three models were 199.0 KiB/s for *Ideal*, 0.1 KiB/s for *TorFlow*, and 30.1 KiB/s for *PeerFlow* in the median, and 960.3 KiB/s for *Ideal*, 729.5 KiB/s for *TorFlow*, and 683.0 KiB/s for *PeerFlow* in the third quartile. Each per-experiment distribution was similar to the combined distribution with a maximum KS statistic of 0.0604.

Given our definition of relay goodput, we define aggregate load L at time T as the sum of all relay goodputs, *i.e.*, $L(T) = \sum_i G_i(T)$. We compute L for all $t \in T$ (every second), and plot the cumulative distribution over all seconds in all experiments for each model. The results are shown in Figure 5b. The median and standard deviation of aggregate goodput are 483.7 and 10.2 MiB/s for the *Ideal* model, 437.9 and 11.3 MiB/s for the *TorFlow* model, and 427.4 and 11.8 MiB/s for the *PeerFlow* model. Some per-experiment distributions differed significantly from the combined distribution with a maximum KS statistic of 0.393. Given these results, we conclude that PeerFlow does not dramatically reduce relay goodput compared to the *TorFlow* model, and that optimizing PeerFlow may be able to increase aggregate relay goodput to be closer to *Ideal*.

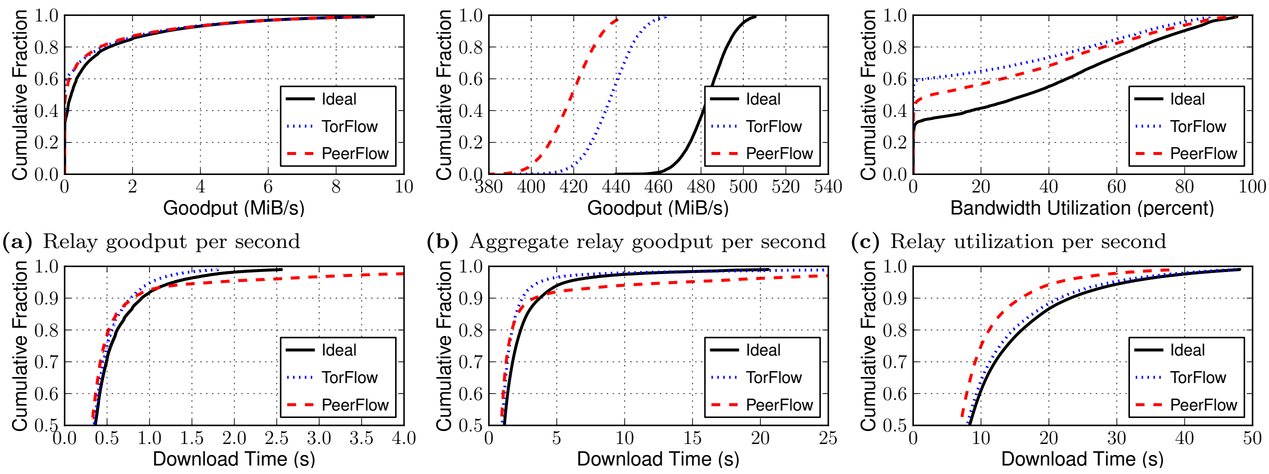
We also consider network utilization as a load-balancing metric, which demonstrates how well the algorithms in our load-balancing models use the available network resources. If relay R_i has goodput $G_i(T)$ and capacity C_i, we define its utilization $U_i(T)$ at time T as $U_i(T) = G_i(T)/C_i$. We compute utilization for each relay each second, and show the distribution of utilization rates over all experiments in Figure 5c. Our results show that 75% of relays had a utilization of 43% or less in *TorFlow*, 49% or less in *PeerFlow*, and 61% or less in *Ideal*. Each per-experiment distribution was similar to the combined distribution with a maximum KS statistic of 0.0628. We thus conclude that while *PeerFlow* improves utilization over *TorFlow*, further improvements may still be possible.

7.3 Client Performance

Another goal of load-balancing algorithms is to minimize client performance bottlenecks. We use file down-

[2] Our Shadow TorFlow plug-in contains 2128 lines of code

[3] Our PeerFlow prototype contains 1222 lines of code



(a) Relay goodput per second **(b)** Aggregate relay goodput per second **(c)** Relay utilization per second

(d) Time to first byte of download **(e)** Time to last byte of 320KiB download **(f)** Time to last byte of 5MiB download

Fig. 5. Network utilization and client performance for the *Ideal*, *TorFlow*, and *PeerFlow* load-balancing models over all experiments.

load times as a metric for determining how clients will perceive performance changes resulting from using PeerFlow. We consider the time to download the first byte of all files as a general measure of latency (Figure 5d), and the time to download the last byte of 320KiB files (Figure 5e) and 5MiB files (Figure 5f) as measures of performance for *web* or *bulk* type downloads, where the distributions considered are across all 3 experiments. As the distribution of download times is nearly identical for all measurements below the median, we focus on the upper half of the distributions.

With regards to the 320KiB file downloads in Figure 5e, the 3rd quartile download time is highest for *Ideal* at 2.00 seconds, followed by *TorFlow* at 1.51 seconds, and lowest for *PeerFlow* at 1.44 seconds. The maximum download time is just over 60 seconds for all models due to the default 60 second timeout set by the traffic generation tool used in Shadow. These trends are similar for both 320KiB and 5MiB downloads, as well as for the time to first byte metric. Each per-experiment distribution was similar to the combined distribution with a maximum KS statistic of 0.0461.

Given our results, we conclude that the security benefits of PeerFlow may come with a slight reduction in download times across all file sizes when compared to both *TorFlow* and *Ideal*, and a small increase in download time for the smaller first byte and 320KiB metrics for less than 10% of the downloads. We also note that while ideal measurement does result in the highest overall network utilization, as expected, it doesn't yield faster individual downloads, and this unintuitive phenomenon may merit further investigation. Furthermore, probation is not fully implemented in our prototype, and so it is possible that PeerFlow will underestimate relays' weights. We now analyze the extent of such errors in weight calculations in our experiments.

7.4 Consensus Weight Errors

TorFlow and PeerFlow produce relay weights that get added to the Tor network consensus. These weights bias the relays chosen by clients when constructing circuits, thus affecting the distribution of client load. Ignoring additional position weights that are applied based on the ability to serve as a guard or exit relay, the ratio of a given relay weight to the sum of all of the relays' weights roughly approximates the fraction of total network load that will be directed to that relay.

A good load-balancing scheme should distribute load proportional to the real capacities of the relays; in other words, a relay's relative weight should match its relative capacity. We can thus consider the *inaccuracy* of load balancing in Tor by comparing the relays' relative consensus weights to their true relative bandwidths. Let C_i be the bandwidth capacity of relay R_i and its relative capacity be $\mathcal{C}_i = C_i/\sum_j C_j$, and let W_i^j be the weight of R_i in the jth consensus and its relative weight be $\mathcal{W}_i^j = W_i^j/\sum_k W_k^j$. We measure the inaccuracy of the weight of R_i as $|\log_{10}(\mathcal{C}_i/\mathcal{W}_i^j)|$. We also measure the *imprecision* of the weights of R_i across consensuses as their sample standard deviation divided by their sample mean.

Table 5 compares the inaccuracy and imprecision of TorFlow and PeerFlow. For each experiment, the inaccuracy is averaged across all relays and consensuses, and the average and standard deviation across experiments is shown. Similarly, the imprecision results are averaged

	Inaccuracy		Imprecision	
	Avg.	Std. Dev.	Avg.	Std. Dev.
TorFlow	1.02	0.244	0.288	0.213
PeerFlow	0.428	0.0381	0.217	0.0814

Table 5. Weight errors in TorFlow and PeerFlow.

over all relays for each experiment, and the average and standard deviation across experiments is shown. The results show that our PeerFlow prototype was both more accurate and more precise than TorFlow.

8 Speed and Efficiency Analysis

PeerFlow schedules measurement rounds individually for each relay depending on the relay's expected client traffic. With more client traffic, a relay's measurements will more quickly be large relative to the added noise, and so the measurement time can be smaller. These measurement times also affect the bandwidth efficiency of the protocol because they determine how often measurements are sent to the Directory Authorities.

8.1 Speed

In order to estimate relay measurement times when running PeerFlow on the Tor network, we use historical network data from CollecTor [2]. These data include past network consensuses, from which we can determine the types of relays that existed, and extra-info descriptors, from which we can determine the amount of traffic each relay transferred.

We use the consensus from January 31, 2015, at 00:00 and the extra-info descriptors from January 2015 as the basis for our analysis. We estimate the average rate of client traffic transferred by each relay using the byte histories over the hour before the consensus. After excluding relays with empty or incomplete byte histories or with selection probabilities of zero, there are 5917 relays in the network.

We estimate PeerFlow's determination $\bar{\rho}_p^R$ of the bytes relayed by R and observed in position p by taking the minimum of read and written bytes in the byte history and multiplying it by R's relative probability of being selected in each circuit position. We take the measuring guards, measuring middles, and measuring exits to be the largest $\mu = 0.75$ fraction of relays by position-weighted bytes relayed (*i.e.*, for relay R, β^R is multiplied by the position weight for R in the given measuring position). The result is 400 measuring guards, 749 measuring middles, and 145 measuring exits.

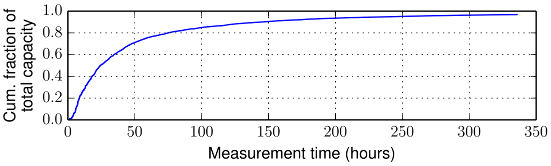


Fig. 6. Measurement times weighted by observed relay capacity

	Number of Relays		Dummy Traffic (KiB/s)	
ϵ_{dec}	Median	Max	Avg. Total	Std. Dev.
0.05	14	75	2.38	0.677
0.25	7	55	1.49	0.573
0.50	4	57	2.05	1.33
0.75	1	17	0.039	0.0142
0.95	1	10	0.00328	0.00385

Table 6. Median and maximum number of relays in probation and the amount of dummy traffic needed to avoid probation

We apply the measurement-time procedure given in §5.4 to estimate what the measurement times of existing Tor relays would be under PeerFlow. Figure 6 shows the resulting distribution of measurement times over the observed bytes transferred. Measurement times below 14 days are shown, which constitute 96.8% of the times by relay capacity. The 25th-percentile measurement time is 11.5 hours, the 50th is 27.7 hours, and the 75th is 70.7 hours. The relays with measurement times above 14 days have low observed capacity, many below Tor's stated requirements of 100 KB/s for Fast relays and 250 KB/s for guards. By raising these requirements by 150%, to 250 KB/s for Fast relays and 625 KB/s for guards, and excluding relays with observed capacities below these requirements, the maximum measurement time can be lowered to 316.4 hours (*i.e.* 13.2 days).

8.2 Efficiency

PeerFlow has bandwidth overhead from sending statistics to the Directory Authorities and from dummy traffic due to relays in probation. Each measuring relay sends data to the Directory Authorities at the end of a measured relay's measurement period. Thus the total measurement traffic to them is $O(mn)$, where m is the number of measuring relays and n is the number of relays. However, each measuring relay sends only the four ρ statistics for each measured relay, and our evaluation shows that measurement periods are generally at least a few hours long. Again using the consensus from January 31, 2015, at 00:00, and assuming that each statistic is 4 bytes, we find that the average rate of upload to each Bandwidth Authority is only 119.6 B/s.

We investigate the dummy traffic overhead by running Shadow experiments over a range of probation-trigger values ϵ_{dec}. Using the PeerFlow model and network described in §7, we ran 3 experiments for each ϵ_{dec}. For $\mu = 0.75$, ϵ_{dec} ranges from 0.90 for a trusted fraction of $\tau = 0.05$ to 0.2 for $\tau = 0.3$. The results in Table 6 show that, out of the 498 total relays, the median number of relays in probation in a given second across all experiments was at most only 14 and that the maximum was at most only 75. Moreover, the results show the total dummy traffic needed to meet expectations and thus avoid probation was at most only 2.38 KiB/s on average over time and across all experiments, with comparable standard deviation across experiments. This amount is very small relative to the median network-wide PeerFlow goodput of 428.0 MiB/s (see §7).

9 Conclusion

Tor's security is vulnerable to an adversary running large relays, and we show that under the Tor's current TorFlow bandwidth-measurement system and the proposed EigenSpeed system an adversary can make small relays appear large, drastically reducing the cost of attack. We present PeerFlow, show how it limits the ability of an adversary to fool Tor about the bandwidth of his relays, and demonstrate that its performance is comparable to Tor's current performance. Possible future improvements to PeerFlow include improving scalability and further improving security to reduce even more the adversary's ability to inflate his relays' weights.

Acknowledgments: We thank the anonymous reviewers for their suggestions and feedback. This work has been partially supported by DARPA and ONR. The views expressed in this work are strictly those of the authors and do not necessarily reflect the official policy or position of DARPA, ONR, or NRL.

References

[1] https://metrics.torproject.org/.

[2] Collector. https://collector.torproject.org/.

[3] Shadow simulator. https://shadow.github.io.

[4] Tor directory protocol, version 3. https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=dir-spec.txt.

[5] Bandwidth scanner spec. https://gitweb.torproject.org/torflow.git/blob_plain/HEAD:/NetworkScanners/BwAuthority/README.spec.txt.

[6] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multicandidate election system. In Ajay D. Kshemkalyani and Nir Shavit, editors, *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, PODC 2001, Newport, Rhode Island, USA, August 26-29, 2001*, pages 274–283. ACM, 2001.

[7] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against Tor. In *ACM WPES*, 2007.

[8] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Trawling for Tor hidden services: Detection, measurement, deanonymization. In *IEEE S&P*, 2013.

[9] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.

[10] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In Kwangjo Kim, editor, *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer, 2001.

[11] Ivan Damgård and Mads Jurik. A length-flexible threshold cryptosystem with applications. In Reihaneh Safavi-Naini and Jennifer Seberry, editors, *Information Security and Privacy, 8th Australasian Conference, ACISP 2003, Wollongong, Australia, July 9-11, 2003, Proceedings*, volume 2727 of *Lecture Notes in Computer Science*, pages 350–364. Springer, 2003.

[12] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *USENIX Security*, 2004.

[13] Cynthia Dwork. Differential privacy. In *International Colloquium on Automata, Languages and Programming*, 2006.

[14] Nathan Evans, Roger Dingledine, and Christian Grothoff. A practical congestion attack on Tor using long paths. In *USENIX Security*, 2009.

[15] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In Yair Frankel, editor, *Financial Cryptography, 4th International Conference, FC 2000 Anguilla, British West Indies, February 20-24, 2000, Proceedings*, volume 1962 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2000.

[16] David Goulet, Aaron Johnson, George Kadianakis, and Karsten Loesing. Hidden-service statistics reported by relays. Technical Report 2015-04-001, The Tor Project, Inc., April 2015.

[17] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. Peerreview: Practical accountability for distributed systems. In *SOSP*, 2007.

[18] Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-Tin. How much anonymity does network latency leak? *TISSEC*, 13(2), February 2010.

[19] Rob Jansen, Kevin Bauer, Nicholas Hopper, and Roger Dingledine. Methodically modeling the Tor network. In *CSET*, 2012.

[20] Rob Jansen, John Geddes, Chris Wacek, Micah Sherr, and Paul Syverson. Never been KIST: Tor's congestion management blossoms with kernel-informed socket transport. In *USENIX Security*, 2014.

[21] Rob Jansen and Nicholas Hopper. Shadow: Running Tor in a box for accurate and efficient experimentation. In *NDSS*, 2012.

[22] Rob Jansen, Aaron Johnson, and Paul Syverson. LIRA: Lightweight incentivized routing for anonymity. In *NDSS*, 2013.

[23] Rob Jansen, Andrew Miller, Paul Syverson, and Bryan Ford. From onions to shallots: Rewarding Tor relays with TEARS. In *HotPETs*, 2014.

[24] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on Tor by realistic adversaries. In *ACM CCS*, 2013.

[25] Ghassan Karame, David Gubler, and Srdjan Capkun. On the security of bottleneck bandwidth estimation techniques. In *SecureComm*. 2009.

[26] Mike Perry. TorFlow: Tor network analysis. In *HotPETs*, 2009.

[27] Robin Snader. *Path Selection for Performance- and Security-Improved Onion Routing*. PhD thesis, U. of I. at Urbana-Champaign, 2009.

[28] Robin Snader and Nikita Borisov. Eigenspeed: Secure peer-to-peer bandwidth evaluation. In *IPTPS*, 2009.

[29] Robin Snader and Nikita Borisov. Improving security and performance in the Tor network through tunable path selection. *TDSC*, 8(5):728–741, September 2011.

[30] R. Suselbeck, G. Schiele, P. Komarnicki, and C. Becker. Efficient bandwidth estimation for peer-to-peer systems. In *IEEE P2P*, 2011.

[31] Fabrice Thill. *Hidden Service Tracking Detection and Bandwidth Cheating in Tor Anonymity Network*. PhD thesis, Univ. Luxembourg, 2014.

[32] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *USENIX Security*, 2014.

[33] Tao Wang and Ian Goldberg. Improved website fingerprinting on Tor. In *ACM WPES*, 2013.

[34] Matthew Wright, Micah Adler, Brian Neil Levine, and Clay Shields. The predecessor attack: An analysis of a threat to anonymous communications systems. *TISSEC*, 4(7):489–522, November 2004.

A Enhanced PeerFlow

While our simulations and analysis show that PeerFlow would improve the security and performance of Tor's load balancing at relatively low cost, we note that some future concerns are not addressed by our basic design. First, revealing the approximate amount of traffic sent between specific pairs of relays increases the information available to an adversary for traffic analysis. In addition, adding noise to traffic statistics makes them less accurate and forces PeerFlow to wait to report on a given relay until enough client traffic is likely to have been relayed. This delays measurement of low-bandwidth relays, and as the network grows the relative weight of each measuring relay will decrease, thus increasing the length of reporting intervals.

Here we describe a modifications to PeerFlow that address these concerns, by cryptographically protecting the privacy of individual measurements. We present these modifications as an enhancement to PeerFlow to leave a core design that is more straightforward for the Tor Project to implement and adopt.

A.1 Encrypted Measurement Aggregation

In order to protect the privacy of individual measuring relay observations, we introduce a new role for the Bandwidth Authorities (§5.7). These authorities will have the role of jointly decrypting bandwidth reports after aggregation; if a majority of the Bandwidth Authorities collude then individual observations can be exposed, but if a majority are honest, then individual observation reports will retain their privacy.

The Bandwidth Authorities cooperate to publish a public key for a *threshold homomorphic tally system*, while separately maintaining shares of the decryption key. In addition to key generation, a threshold homomorphic tally system supports the following operations:

- *Encryption*: Given a public key and t tallies $b_1, b_2, \ldots, b_t$, produce a ciphertext that encodes these counts, providing semantic security.
- *Proving*: Given a public key PK, randomness r, and a ciphertext that encrypts a tally, produce a publicly-verifiable proof that the ciphertext is well-formed, *i.e.* was produced by computing $E_{PK}(0, \ldots, 1, \ldots, 0; r)$.
- *Aggregation*: Given ciphertexts c and c' encoding tallies $b_1, \ldots, b_t$ and $b'_1, \ldots, b'_t$ and weights w and w', produce a ciphertext c that encodes $wc_1 + w'c'_1, \ldots, wc_t + w'c'_t$.
- *Decryption*: Given ciphertext c and decryption key share s, produce a decryption share d along with a publicly-verifiable proof of correctness. The shares should be publicly combinable to produce the joint decryption. This verifiability allows PeerFlow to maintain auditability while increasing privacy.

We describe two possible implementations of these operations in §A.2.1 and A.2.2.

Given the encrypted tally system, we modify how measuring relays report their observations. For each relay R, we partition the range $((1-\delta)\rho_p^R, (1+\delta)\rho_p^R)$ into t equally-sized buckets $[b_1, b_2), [b_2, b_3), \ldots, [b_{t-1}, b_t)$, where δ is a maximum allowed relative capacity change. Each measuring relay in position p then computes the observed value ρ_p^R (without noise) as in §5.1, and chooses the bucket $b \in \{1, \ldots, t\}$ containing ρ_p^R. The measuring relay R' then encrypts a "vote" for this bucket, and submits this encrypted tally $c^{R'}$ to the

Bandwidth Authorities along with a proof of well-formedness.

Finally, the Bandwidth Authorities use the homomorphic properties of submitted ballots to combine the tallies, weighted by measuring relays' voting weights, obtaining a final tally for each relay observation. The Bandwidth Authorities decrypt this aggregate, providing proofs of correct decryption. The Bandwidth Authorities then compute the observation for relay R, $\bar{\rho}_p^R$, as the mean value of the votes after trimming λ voting weight from the smallest and largest buckets. They send this value to the Directory Authorities.

A.2 Example Threshold Homomorphic Tally Schemes

A.2.1 Paillier-based scheme

Baudron et al. [6] describe a homomorphic tally scheme based on the Paillier cryptosystem; the scheme can be modified for PeerFlow in a straightforward way. The Bandwidth Authorities engage in a distributed threshold Paillier key generation algorithm [10, 11, 15], resulting in a Paillier public key N with generator g. Using the network consensus, all measuring relays can agree on a value M which is greater than the sum S of all voting weights, for example $M = 2^{\lceil \log_2 S \rceil}$. Then to encrypt a vote for bucket i, a measuring relay chooses $r \in_R \mathbb{Z}_N^*$ and computes $E_N(b_i) = g^{M^i} r^N \bmod N^2$.[4]

Note that in this scheme, if the sum of voting weight given to bucket i is v_i, we have $E_N(b_i)^{v_i} = E_N(v_i M^i)$ and $\prod_i E_N(v_i M^i) = E_N(\sum_i v_i M^i)$, so as long as we have all $v_i < M$, we can recover weights $v_1, ...v_t$ by representing the result of decryption in base M (and this will be particularly efficient if $M = 2^k$ for some k).

Proving in zero-knowledge that a ciphertext $c = E_N(M^i)$ correctly encrypts a particular bucket value M^i can be accomplished by proving knowledge of an N-th root of c/g^{M^i} using the Guillou-Quisquater scheme, which requires the prover to send two elements of $\mathbb{Z}_{N^2}$ and a 2ℓ-bit challenge for soundness level $2^{-\ell}$; this can be extended using the standard techniques of Cramer *et al.* [9] to prove that $\bigvee_{i=1}^t c = E_N(M^i)$ and converted into a noninteractive proof using the Fiat-Shamir heuristic. All together, this method of proving that a ciphertext is correct produces a proof of length $2t(2\log_2 N + \ell)$ bits, and can be verified with t modular exponentiations in $\mathbb{Z}_{N^2}$.

Proofs of correct decryption are provided by each Bandwidth Authority individually, following the protocol described in [10, 11, 15].

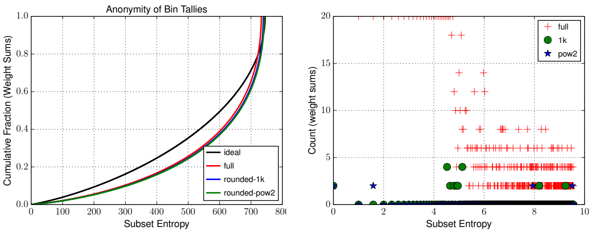
A.2.2 Additive Elgamal-based scheme

A tally scheme can also be implemented using additive Elgamal encryption over any group where the Decisional Diffie-Hellman assumption is hard. In this case, the bandwidth authorities engage in a distributed key generation protocol to produce t public keys $\{h_i = g_i^x\}_{i=1}^t$ with generators $g_0, g_1, \dots, g_t$ of prime order p whose respective discrete logarithms are unknown. To encrypt a vote for bucket b, a measuring relay chooses $r \in_R \mathbb{Z}_p$, and computes $c_0 = g_0^r$, $c_b = h_b^r g_0$, and $c_j = h_j^r$ for $j \neq b$; the ciphertext becomes the list $c_0, c_1, \dots, c_t$. Note that in this scheme, raising all elements of a ciphertext to a scalar power v scales the tally for bucket b by v, and elementwise multiplication adds the tallies for all buckets. After decrypting the elements of a ciphertext using standard threshold Elgamal, the individual tallies are recovered in $O(\sqrt{\sum_i v_i})$ time using a standard short discrete logarithm algorithm.

Proving that a ciphertext c encrypts a vote for a single bucket b in honest-verifier zero knowledge could be accomplished as follows: the verifier chooses a random vector $\langle x_1, \dots, x_t \rangle \in_R \mathbb{Z}_p^t$; then both sides compute $X = \sum_i x_i$, $H = \prod_i h_i^{x_i}$, and $C_b = g_0^{-x_b} \prod_i c_i^{x_i}$. Finally, the verifier and prover engage in a standard proof of knowledge of equality of discrete logarithms, $\log_{g_0} c_0 = \log_H C_b$, which requires the prover to send a group element and an element of $\mathbb{Z}_p$ and the verifier to send an ℓ-bit challenge. Applying the standard disjunction technique to the t possible buckets allows constructing a proof that c encrypts a single vote for one bucket $b \in \{1, \dots, t\}$, and applying the Fiat-Shamir heuristic results in a noninteractive proof of length t group elements plus t elements of $\mathbb{Z}_p$ and t short exponents.

Privacy Analysis: Under the assumption that the majority of Bandwidth Authorities are honest, this scheme ensures that no information is leaked about the individual measurements reported by each measuring relay beyond what is leaked by the voting-weighted sum released for a given measurement period; this follows from the semantic security of the tally encryption schemes.

[4] If $M^t > N$, the scheme can use the Damgård-Jurik extension to work modulo N^{d+1}, where d is the minimal integer such that $M^t < N^d$.



(a) **(b)**

Fig. 7. The subset entropy of four voting-weight rounding schemes applied to the top 750 bandwidth weights from a recent Tor consensus: *full* is no rounding, *ideal* assigns all weights to 1, *rounded-1k* rounds to the nearest 1000, *rounded-pow2* rounds up to the nearest power of 2. 7a Cumulative frequencies of subset anonymity; 7b counts of low-anonymity subsets (capped at 20 to show detail).

Here we analyze the amount of information that could be leaked by this aggregate result.

The residual leakage stems from the fact that few relays have identical weights in the current Tor network, so many small subsets of relays could have distinctive weight sums. As an example, we consider the bandwidth weights of the top 750 relays in the Tor network as of February 9, 2015 (750 is just over the number of measuring relays we used in our analysis §8.1). Among these relays, there are 138 possible sums that can only arise from a single subset of the weights, and 2456 possible sums that can arise from fewer than 256 different relay subsets.[5]

However, this leakage can be mitigated to a large extent by one of two schemes for rounding of voting weights. In the *Rounded-1K* scheme, each measuring relay's weight is divided by 1000 and rounded. Continuing with the example above, we find that the 750th highest-weighted relay on Febuary 9, 2015 had a weight of 8160, which under this scheme would be assigned voting weight 8, while the highest-weighted relay had a weight of 334000, and would be assigned voting weight 334. In the *Rounded-Pow2* scheme, each measuring relay's weight is rounded to the next power of 2 and scaled by the voting weight of the lowest-weighted measuring relay. Thus the lowest-weighted measuring relay in the previous example would have its weight rounded to 8192 and then scaled to 1, while the highest-weighted relay's voting weight would be rounded to $2^{19} = 524288$ and scaled to 64. Figure 7 summarizes the impact of these rounding schemes on the anonymity of weight sums: overall, most relay subsets have high subset entropy (defined as $SE(R) = \log_2 |\{S \subseteq [n] : \sum_{i \in S} w_i = \sum_{i \in R} w_i\}|$), as seen in Figure 7a. As Figure 7b shows, however, both rounding schemes significantly reduce the number of subsets with low subset entropy, with *Rounded-Pow2* producing the fewest low-entropy subsets.

We note that in the worst case, the *Rounded-Pow2* scheme may inflate the fraction of voting weight controlled by an adversary by nearly a factor of 2, whereas the worst case for the *Rounded-1K* scheme is $1 + \frac{1}{2v_{\min}+1}$, where $v_{\min} \geq 1$ is the lowest post-rounding voting weight among measuring relays; in our example $v_{\min} = 8$, so the worst-case inflation factor as a result of rounding was 1.059. If we instead consider an adversary that compromises existing relays in our example, the highest relative inflation encountered under the *Rounded-Pow2* scheme was 1.366, while the highest relative inflation under the *Rounded-1K* scheme was 1.014; the respective 90th percentiles were 1.287 and 1.014.

B Proofs of Theorems

Lemma 1. *If $\alpha < \lambda$, then over the random noise values*

$$\mathbb{E}[\rho^A] \leq ((1 - \lambda - \alpha)\mu)^{-1}\, \mathbb{E}\left[\max(S_1, \beta^A + S_2)\right].$$

Proof. The Directory Authorities infer the number of bytes transferred by A as the maximum of inferred bytes observed on the client and destination sides: $\rho^A = \max(\rho^A_c, \rho^A_d)$. The inferred number of client-side bytes is the sum of the inferred number of guard-observed and middle-client-side bytes: $\rho^A_c = \bar{\rho}^A_g + \bar{\rho}^A_{mc}$. The inferred number of destination-side bytes is defined similarly: $\rho^A_d = \bar{\rho}^A_e + \bar{\rho}^A_{md}$. Let β^A_p be the number of bytes sent or received by A with a measuring relay in position $p \in \{g, mc, md, e\}$. We will show a bound on each $\bar{\rho}^A_p$ in terms of β^A_p that will combine to bound both ρ^A_c and ρ^A_d and then ρ^A as well.

$\bar{\rho}^A_p$ is determined from the values $\beta^{R_i A}_p$, which are the number of bytes A sent or received from each measuring relay $R_i \in \mathcal{M}_p$ while R_i was in position p in the circuit. Each R_i uses $\beta^{R_i A}_p$ to produce an independent estimate of the total bytes $\rho^{R_i A}_p$ by first adding random noise $N^{R_i A}_p \sim \mathsf{Lap}(\delta_{\text{noise}}/\epsilon_{\text{noise}})$ and then dividing by its selection probability $q^{R_i}_p$, that is, $\rho^{R_i A}_p = (\beta^{R_i A}_p + N^{R_i A}_p)/q^{R_i}_p$. $\bar{\rho}^A_p$ is the aggregate of these estimates after trimming the largest and smallest fraction λ by voting weight. Let $i_{p,j}$ be the index of the jth

[5] To count possible subset sums, we modified the standard pseudopolynomial-time dynamic programming algorithm for subset sum, which runs in time $O(mS)$ for m items with total weight S

largest ρ statistic. Let i_p^1 and i_p^2 be the indices of the first and last untrimmed statistics, respectively. Let $\mathcal{A}$ be the set of indices of the adversary's relays. To extend the measuring relay notation to directional positions, let $\mathcal{M}_{mc} = \mathcal{M}_{md} = \mathcal{M}_m$. We bound the value of $\bar{\rho}_p^A$ as follows:

$$\bar{\rho}_p^A = \frac{\sum_{i_p^1 \leq j \leq i_p^2} \rho_p^{R_{i_{p,j}}A} q_p^{R_{i_{p,j}}}}{\sum_{i_p^1 \leq j \leq i_p^2} q_p^{R_{i_{p,j}}}} \tag{1}$$

$$= \frac{\sum_{i_p^1 \leq j \leq i_p^2} \beta_p^{R_{i_{p,j}}A} + N_p^{R_{i_{p,j}}A}}{\sum_{i_p^1 \leq j \leq i_p^2} q_p^{R_{i_{p,j}}}} \tag{2}$$

$$\leq \frac{\sum_{i_p^1 \leq j \wedge j \notin \mathcal{A}} \beta_p^{R_{i_{p,j}}A} + N_p^{R_{i_{p,j}}A}}{\sum_{i_p^1 \leq j \wedge j \notin \mathcal{A}} q_p^{R_{i_{p,j}}}} \tag{3}$$

$$\leq \frac{\beta_p^A + \sum_{i_p^1 \leq j \wedge j \notin \mathcal{A}} N_p^{R_{i_{p,j}}A}}{(1-\lambda-\alpha)\mu} \tag{4}$$

The inequality in line 3 holds because $j > j'$ implies that $\rho_p^{R_{i_{p,j}}A} \geq \rho_p^{R_{i_{p,j'}}A}$ and also any untrimmed voting weight from adversarial relays is replaced by the same amount of voting weight from honest relays with larger ρ statistics. The inequality in line 4 holds because the voting weight of honest relays with ρ statistics above the lower trimmed fraction is at least $1-\lambda-\alpha$, and voting weights are updated to maintain that the selection probability of any subset with voting weight at least $1-2\lambda$ has total selection probability of at least μ of its voting weight.

Let $S_p = \sum_{i_p^1 \leq j \wedge j \notin \mathcal{A}} N_p^{R_{i_{p,j}}A}$. Then, using our bound on $\bar{\rho}_p^A$, we can see that

$$\rho_c^A = \bar{\rho}_g^A + \bar{\rho}_{mc}^A \tag{5}$$

$$\leq \frac{\beta_g^A + \beta_{mc}^A + S_g + S_{mc}}{(1-\lambda-\alpha)\mu}. \tag{6}$$

By similar arguments, an analogous bound holds for ρ_d^A.

ρ^A is the maximum of ρ_c^A and ρ_d^A. Thus, by the preceding inequalities on those values (*e.g.* Inequality 6),

$$\begin{aligned}\mathbb{E}[\rho^A] &= \mathbb{E}[\max(\rho_c^A, \rho_d^A)] \\ &\leq \mathbb{E}\Bigg[\frac{1}{(1-\lambda-\alpha)\mu} \max\big(\beta_g^A + \beta_{mc}^A + S_g + S_{mc}, \\ &\qquad \beta_e^A + \beta_{md}^A + S_e + S_{md}\big)\Bigg]. \end{aligned} \tag{7}$$

This expectation is over all possible values of the noise variables sums (*i.e.* the S_p). Moreover, each noise variable is symmetric. Therefore, if we add an additional independent noise variable into one of those sums, for every value of the added variable that decreases the maximum there is an equally-probable value of the added variable that increases it by the same amount (the reverse is not true because we are taking the maximum with another value). Therefore, adding in additional independent noise variables only increases the expectation on the right-hand side of Inequality 7. Let $n_{\text{msr}} = \max(|\mathcal{M}_g| + |\mathcal{M}_m|, |\mathcal{M}_e| + |\mathcal{M}_m|)$, and let $S_j = \sum_{1 \leq i \leq n_{\text{msr}}} N_i^j$, $j \in \{1,2\}$, where each N_i^j is an independent random variable with distribution $\mathsf{Lap}(\delta_{\mathsf{noise}}/\epsilon_{\mathsf{noise}})$). Then, by the foregoing argument,

$$\begin{aligned}\mathbb{E}[\rho^A] \leq \mathbb{E}\big[((1-\lambda-\alpha)\mu)^{-1} \max\big(&\beta_g^A + \beta_{mc}^A + S_1, \\ &\beta_e^A + \beta_{md}^A + S_2\big)\big]. \end{aligned} \tag{8}$$

For a fixed bandwidth budget β^A, the right-hand side of inequality 8 can only be increased by increasing the larger of $\beta_g^A + \beta_{mc}^A$ and $\beta_e^A + \beta_{md}^A$ and decreasing the smaller. The follows from the fact that S_1 and S_2 are independently and identically distributed, and so every case in which shifting the β_p^A in this way reduces the maximum reduced corresponds to a unique and equally-probably case in which the maximum is increased by an equal or larger amount (simply swap the values of S_1 and S_2. Therefore, for fixed β^A, the right-hand side of Inequality 8 is maximized by setting $\beta_g^A + \beta_{mc}^A$ to β^A and $\beta_g^A + \beta_{mc}^A$ to 0 (vice versa will maximize it as well). Therefore,

$$\mathbb{E}[\rho^A] \leq \frac{1}{(1-\lambda-\alpha)\mu} \mathbb{E}\left[\max(S_1, \beta^A + S_2)\right],$$

which was the statement to be proved. □

Lemma 2. *If $\alpha < \lambda$, then, for all $R \notin \mathcal{A}$, $\rho^R \geq \gamma'\beta^R/2$.*

Proof. We can assume that the adversarial relays $A \in \mathcal{A}$ send values ρ_p^{RA} that are lower than all other values $\rho_p^{RR'}$, because doing so can only reduce ρ^R. Moreover, $\alpha < \lambda$, and so we can assume the set of measurements in the trimmed sum are from some set of honest relays of voting weight $1-2\lambda$. The aggregate of these measurements in a given position is at least $\gamma'\beta_p^R$. By the requirement that a relay will not act in both the guard and exit positions during a measurement period, R sends all bytes from either a client-side or destination-side position. Furthermore, R sends at least as many bytes as it receives as it never drops data. Therefore, either $\beta_g^R + \beta_{mc}^R$ or $\beta_e^R + \beta_{md}^R$ is at least $\beta^R/2$, and so

$$\begin{aligned}\rho^R &= \max(\bar{\rho}_g^R + \bar{\rho}_{mc}^R, \bar{\rho}_e^R + \bar{\rho}_{md}^R) \\ &\geq \gamma' \max(\beta_g^R + \beta_{mc}^R, \beta_e^R + \beta_{md}^R) \\ &\geq \gamma'\beta^R/2.\end{aligned}$$

□

To prove Theorem 1, we first quantify as ν_{loss} the loss in accuracy of ρ^R due taking the maximum of noisy values, which is limited to ϵ_{loss} (see § 5.4). ν_{loss} is defined as the smallest value such that for all β large enough that $\mathbb{E}[\max(S_1, \beta+S_2)]/((1-2\lambda)\mu) \geq (1-\epsilon_{\text{dec}})\rho_0^A t^A/t_0^A$, $\beta/\mathbb{E}[\max(S_1, \beta + S_2)] \geq 1 - \nu_{\text{loss}}$.

Theorem 1. *For $\alpha < \lambda$ and over the random noise values, if $\beta^A < (1-\lambda-\alpha)(1-\epsilon_{dec})(1-\epsilon_{loss})\mu\rho_0^A t^A/t_0^A$, then $\mathbb{E}[\rho^A]$ puts A into probation, and otherwise*

$$\frac{\mathbb{E}[\omega^A]}{\sum_R \omega^R} \leq \left[\frac{2\gamma'(1+\epsilon_{inc})}{(1-\epsilon_{loss})(1-\lambda-\alpha)\mu}\right] \frac{(\beta^A/t^A)}{\sum_R(\beta^R/t^R)}.$$

Proof. If $\beta^A < (1-\lambda-\alpha)(1-\epsilon_{\text{dec}})(1-\epsilon_{\text{loss}})\mu\rho_0^A t^A/t_0^A$, then, by Lemma 1 and the definition of ϵ_{loss}, $\mathbb{E}[\rho^A] < (1-\epsilon_{\text{dec}})\rho_0^A t^A/t_0^A$. ω^A is adjusted to keep ρ^A no less than $(1-\epsilon_{\text{dec}})\eta^R$, and thus the expected value of ρ^A would put A into probation.

Otherwise, it must be that $\beta^A/\mathbb{E}[\max(S_1, \beta^A + S_2)] \geq 1 - \epsilon_{\text{loss}}$ by the definition of ϵ_{loss}. Therefore, by Lemma 1, $\mathbb{E}[\rho^A] \leq \beta^A/((1-\epsilon_{\text{loss}})(1-\lambda-\alpha)\mu)$. Then we can use Lemma 2 and the fact that ω^A is set to at most $(1+\epsilon_{\text{inc}})\rho^A/t^A$ to obtain the theorem. □