FlashFlow: A Secure Speed Test for Tor

Matthew Traudt
U.S. Naval Research Laboratory
matthew.traudt@nrl.navy.mil

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

Aaron Johnson
U.S. Naval Research Laboratory
aaron.m.johnson@nrl.navy.mil

***Abstract*—The Tor network uses a measurement system called TorFlow to estimate its relays' forwarding capacity and to balance traffic among them. This system has been shown to be vulnerable to adversarial manipulation, and inaccuracies even in benign circumstances have long been observed. To solve the issues with security and accuracy, we present FlashFlow, a system to measure the capacity of Tor relays. Our analysis shows that FlashFlow limits a malicious relay to obtaining a capacity estimate at most 1.33 times its true capacity. Through realistic Internet experiments, we find that FlashFlow measures relay capacity with $\geq$89% accuracy 95% of the time. Through simulation, we find that FlashFlow can measure the entire Tor network in less than 5 hours using 3 measurers with 1 Gbit/s of bandwidth each. Performance simulations using FlashFlow for load balancing shows that, compared to TorFlow, network weight error decreases by 86%, while the median of 50 KiB, 1 MiB, and 5 MiB transfer times decreases by 15%, 29%, and 37%, respectively. Moreover, FlashFlow yields more consistent client performance: the median rate of transfer timeouts decreases by 100%, while the standard deviation of 50 KiB, 1 MiB, and 5 MiB transfer times decreases by 55%, 61%, and 41%, respectively. We also find that the performance improvements increase relative to TorFlow as the total client-traffic load increases, demonstrating that FlashFlow is better suited to supporting network growth.**

I. Introduction

Tor [16] is the most popular system on the Internet for anonymous communication. Tor is currently comprised of about 6,500 geographically diverse volunteer-operated *relays* transferring nearly 200 Gbit/s in aggregate traffic from between 2 million [7] and 8 million [26] daily active users. Tor has seen significant growth recently, nearly doubling the amount of traffic it forwards in the last two years [7].

Tor uses a load-balancing system called TorFlow [28] to balance load from its millions of users across its thousands of relays. The goal of TorFlow is to equalize Tor performance across all clients regardless of which relays they use. It receives bandwidth self-measurements from relays and also makes active measurements of download speeds through each relay. It then computes per-relay weights by multiplying the self-measured bandwidths by their actively measured speed relative to the average (see § II). Clients choose relays for their circuits with probabilities proportional to these weights.

Previous work has shown that TorFlow is insecure. A malicious relay can increase the fraction of traffic it can observe beyond the fraction of Tor bandwidth it provides [10, 12, 23, 34], increasing its ability to deanonymize Tor users using a traffic correlation attack [24, 27]. A main reason for TorFlow's vulnerability is that it trusts relays to accurately self-report their observed capacity. Also, TorFlow's active measurements are supposed to occur concurrently with normal client traffic, but a malicious relay can detect its measurement and throttle client traffic to increase its measured speed.

In addition to its insecurity, TorFlow has been observed to inaccurately measure the capacity of many relays [14]. It can take weeks for a relay's true forwarding capacity to be determined, and for some relays that never occurs [1]. Moreover, a relay's weight can vary significantly even while its capacity stays the same [5]. In part due to these problems, Tor is reimplementing the TorFlow tool using more recent libraries and language standards [25] but without changing the fundamental aspects of the original design.

Unfortunately, TorFlow's bias and variance are fundamental to its design rather than to a particular implementation. TorFlow ultimately relies on the existence of sufficient client traffic to fill a relay's capacity, as the active weight adjustments are limited to correcting disproportionate allocations of that client traffic. Therefore, relay capacities will be systematically underestimated unless the network has a very high utilization rate. Moreover, the active adjustments depend on the available capacity at a relay during a short test download, which varies based on transient and uncontrollable client demands.

We present FlashFlow to solve these problems. FlashFlow is a system designed to securely, accurately, and quickly measure the capacity of relays in the Tor network. In addition to providing weights for load balancing, the capacity measurements allow Tor to accurately assess the network's resources and plan for the future.

The need for security heavily influences the design choices of FlashFlow. We cannot make use of measurement approaches that are vulnerable to manipulation, such as packet pairs [29]. Previously proposed systems attempt to measure Tor surreptitiously [9, 28] or to securely aggregate passive observations made by many relays [23, 32]. FlashFlow takes a new approach to this problem by using separate measurement teams that attempt to actively utilize the *full* capacity of relays. This approach improves security as it requires the direct demonstration of a relay's capacity rather than relying on an indirect measurement that may be falsifiable. It also yields higher accuracy, as the traffic is actively generated to determine a relay's limit, with the normal client traffic carefully reduced to limit its impact on the result without excessively reducing client performance. FlashFlow additionally aggregates results from multiple measurers in order to accurately measure even the highest-capacity relays in Tor.

We implement FlashFlow and conduct extensive experiments in a lab setting, on the Internet, and in simulation. With our suggested parameter settings, FlashFlow limits a malicious relay to obtaining a capacity estimate of *at most* $1.33\times$ its true capacity ($177\times$ was demonstrated for TorFlow [23]). Through Internet experiments across a range of geographic locations, we find that FlashFlow is able to measure a target relay with a capacity ranging from 10 Mbit/s to 1 Gbit/s to within 11% of ground truth in 30 seconds 95% of the time (and within 20% of ground truth 99.8% of the time). Through simulation, we find that FlashFlow can measure the entire Tor network in less than 5 hours using 3 measurers each with 1 Gbit/s of available bandwidth. Through private Tor network simulations in Shadow, we find that FlashFlow reduces network weight error by 86%. The resulting improvement in load balancing reduces transfer times for *all* tested transfer sizes: the median of 50 KiB, 1 MiB, and 5 MiB transfer times decreases by 15%, 29%, and 37%, respectively. FlashFlow also yields more *consistent* client performance: the median rate of transfer timeouts decreases by 100%, while the standard deviation of 50 KiB, 1 MiB, and 5 MiB transfer times decreases by 55%, 61%, and 41%, respectively. Finally, we find that the performance improvements increase further as the total client-traffic load increases, demonstrating that FlashFlow is better suited to supporting Tor network growth than is TorFlow.

Additionally, we are working to transition FlashFlow: we wrote a Tor proposal [35], released our open-source code[1], and actively discuss transition plans with Tor developers.

II. Background

Overview: As of August 2019, the Tor network includes about 6,500 *relays* that forward a combined 200 Gbit/s of Tor traffic, and 9 *Directory Authorities* (DirAuths) that act as trust anchors for the distribution of network information to Tor users. When new relays join the network, they publish their public key and network address to the DirAuths, who then verify reachability and validate Tor protocol support. A voting process occurs every hour, after which the DirAuths add valid relays to a *network consensus* document signed by all authorities and distributed to all Tor clients and relays. The consensus document stores information about all available relays and is required for new clients to use Tor. New relays that appear in a consensus are not used until their performance has been measured by a majority of the 6 *Bandwidth Authorities* (BWAuths) that participate in Tor's load balancing system.

TorFlow: Each Bandwidth Authority runs the TorFlow [28] relay-measurement tool to measure the relative performance of relays in the Tor network over time. TorFlow conducts performance measurements of Tor relays by creating 2-hop Tor circuits through them and downloading one of a set of 13 fixed-sized files (2^i KiB for $i \in \{4, \ldots, 16\}$) from a known destination through each circuit. Every hour, TorFlow aggregates the latest relay measurements and produces a load-balancing *weight* for each relay.

[1]https://gitlab.torproject.org/pastly/flashflow

To assist in balancing load across relays, TorFlow attempts to produce larger weights for relays that can better handle Tor traffic. To compute the weights, TorFlow relies on two data sources. First, TorFlow uses each relay's self-reported bandwidth information that is published every 18 hours in a *server descriptor*. This information includes any rate limit set by the relay (e.g., with the BandwidthRate and BandwidthBurst options [8]), as well as its *observed bandwidth*, which is the highest Tor throughput that the relay was able to sustain for any 10-second period during the last 5 days [15, §2.1.1]. From this information, TorFlow computes the relay's *advertised bandwidth* as the minimum of the observed bandwidth and any rate limit set by the relay. Second, TorFlow uses the results of its own measurements to compute for each relay a ratio of the measurement speed of the relay to the mean measurement speed of all relays in the network. Finally, TorFlow computes a weight for each relay by multiplying the computed speed ratio for that relay by its advertised bandwidth.

Load Balancing and Circuits: The TorFlow weights are collected and reported to the Directory Authorities, added to the following network consensus, and distributed to clients. Tor clients then use the normalized weights as probabilities when selecting relays for their paths through the Tor network in an attempt to balance user load across relays. To use Tor, a client creates a *circuit* through a sequence of three relays, over which a TCP connection can then be made to any Internet host. Communication *cells* of a fixed 514-byte length are sent through the circuit and are encrypted (or decrypted, depending on the direction) by each relay using a key exchanged with the client during circuit construction.

Terms: We use the term *throughput* to mean an amount of traffic that an application or a segment of the network stack (e.g., TCP) has been measured to have forwarded (i.e. received and then sent). We use the term *capacity* to mean the maximum throughput that an application or network segment can handle. Thus a *Tor throughput* is an amount of traffic that a Tor process has been measured to have forwarded, potentially as an estimate of *Tor capacity*. Tor throughput includes cell payloads and headers but excludes TCP, IP, and other network packet headers. Finally, *Tor ground truth* is an estimate of Tor capacity experimentally determined by sending load from increasing numbers of simulated clients and measuring them at the relay. Tor ground truth measurements are accurate but expensive and require trust in the relay.

III. FlashFlow Design

We now present the design for FlashFlow, a system to measure the capacity of Tor relays. This section is organized as follows: Section III-A provides an overview of FlashFlow's design. Section III-B describes the process of performing a single measurement. Section III-C describes how a relay can be measured multiple times to obtain an accurate measurement and how old and new relays are measured differently. Finally, Section III-D describes how a measurement schedule for the entire Tor network is created.

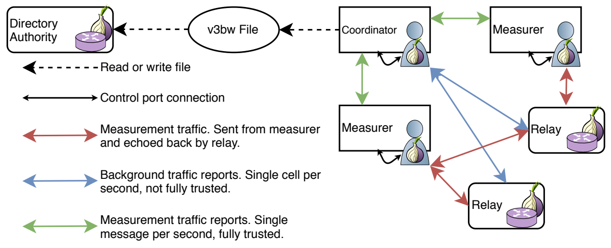

Directory Authority
v3bw File
Coordinator
Measurer
Read or write file
Control port connection
Relay
Measurer
Measurement traffic. Sent from measurer and echoed back by relay.
Background traffic reports. Single cell per second, not fully trusted.
Relay
Measurement traffic reports. Single message per second, fully trusted.

Fig. 1: Example of one FlashFlow team as it measures two relays. The BWAuth is the coordinator and its measurement team. It produces a v3bw file for a DirAuth to use for relay weight voting.

A. *Overview*

Design Criteria: We have designed FlashFlow according to the following design criteria.

- *Efficiency:* We should not require too many resources (most prominently bandwidth) of the operators.
- *Accuracy:* The measurements we conduct should allow us to estimate relay capacity with high accuracy.
- *Speed:* We should quickly measure single relays (e.g., in seconds) and the entire network (e.g., in hours).
- *Security:* We should provide strong security guarantees that limit the amount that a malicious relay can cheat to cause an artificial increase in its weight.
- *Performance:* We should limit the impact of our measurements on client performance, and the weights we produce should improve performance due to better load balancing.

Key Idea: The key technique behind FlashFlow is to actively measure the *full capacity* of Tor relays using a team of multiple measurement hosts. This approach improves security over prior approaches, as relays must demonstrate their true capacity, a process that cannot be faked. It also improves accuracy, as the measurement does not depend on background traffic or on other relays.

Setup: A FlashFlow team consists of two parts: a *measurement team* consisting of one or more *measurers*, and one *coordinator*. The FlashFlow team constitutes a BWAuth; it can be operated entirely by a DirAuth, using as many hosts as necessary to meet the bandwidth requirements, or by an entity the DirAuth trusts that has spare resources to use for relay measurement (e.g. an ISP, hosting provider, or non-profit like torservers.net). Figure 1 shows a FlashFlow team with two measurers as they measure two Tor relays.

The measurers run on hosts whose resources are dedicated to the measurement process. The measurers cooperatively measure relays, so the primary requirement for them is that they collectively have sufficient network capacity to measure all Tor relays. A measurement team is considered to have sufficient capacity if the sum of capacities over all measurers is at least some constant factor f (see § V) times the highest Tor capacity among relays. FlashFlow is designed to achieve accurate measurements given sufficient network capacity, regardless of network latency.

The coordinator controls the measurement team, determines the measurement *schedule*, and aggregates the results. A measurement schedule is created for each measurement *period*, which divides time into constant-length intervals. Multiple BWAuths, each with its own measurement team, independently run FlashFlow. Each BWAuth separately measures each relay during a period. The coordinator records the results as weights in a *v3bw file*, a format already used by DirAuths. Then standard Tor processes take over: the DirAuth reads the v3bw files produced by its BWAuth and votes with other DirAuths for consensus on relay weights every hour.

Trust and Diversity: As in Tor currently, each DirAuth chooses to trust some BWAuth, and the DirAuths place the median of their measurements in the consensus. Thus the trust assumption in FlashFlow is that a majority of DirAuths (and their associated BWAuths) are honest. We recommend and expect the simple case that each DirAuth trusts its own BWAuth; thus FlashFlow requires an honest majority among the BWAuths. Measurement teams are deployed across network locations reflecting the diversity of Tor clients and relays to mitigate unrepresentative measurements resulting from unusually high or low network capacity between a relay and a measurement team (e.g., due to existing in the same data center).

B. *A Single Measurement*

A BWAuth initiates a single measurement by creating an authenticated connection from its coordinator to each measurer (the green connections in Figure 1) and to the target relay (blue connections). Authentication is performed using the public key of the coordinator, which we assume is distributed in the Tor network consensus. The coordinator sends the target the public keys of each measurer involved in the measurement. While connected, the measurers accept instructions from the coordinator, and the relay accepts authenticated measurement connections (red connections) from the measurers indicated by the coordinator. The relay will only accept connections from a given coordinator and its team once per measurement period.

The coordinator will divide the total resources needed for the measurement across its m measurers $M_1, \ldots, M_m$. It allocates a quantity a_i of the measurement capacity of M_i to the measurement (see § III-C for choosing a_i), where $a_i = 0$ is possible and indicates that M_i does not participate in the measurement. Each measurer M_i starts $k_i = \max(1, c_i/p_i)$ modified Tor processes for the measurement, where c_i is the number of available CPU cores on M_i and p_i is the expected number of parallel measurements in which M_i will participate during the measurement interval. The measurement-traffic rate of the k_i processes thus started for a given measurement is limited by setting the BandwidthRate parameter of each modified Tor process on the measurer to a_i/k_i. A constant total number of TCP sockets s is used across all measurers (the value for s is determined experimentally), and each M_i uses an even share s/m of them, with each measuring process at M_i using $s/(mk_i)$ of the sockets.

Each measuring process creates one TLS connection with the target relay for each of its allocated sockets. Over each such connection, a special measurement circuit is constructed using a new type of circuit-creation cell. A key exchange is performed, but the circuit will not be extended further. All cells received on the circuit by the target relay will be decrypted and then returned to the measurer. The target relay schedules cells on measurement circuits using a separate cell scheduler to ensure high throughput even with fewer sockets than typical for a Tor relay (the existing scheduler [21] is designed for priority scheduling across many sockets [18]). Moreover, the target relay enforces a maximum ratio r between cells sent by the normal scheduler and those overall, and it attempts to send as much normal traffic as possible subject to this maximum. This design provides an accurate measurement while ensuring that normal traffic continues to be relayed.

A relay is measured by a BWAuth during a measurement *slot*. During this time, each measuring process sends measurement cells filled with random bytes over the measurement circuits. The process sends such cells as fast as possible. The target relay decrypts those cells using the circuit key, and then returns them on the circuit. Note that both the measurer and the target perform TLS encryption and decryption, but the target alone performs Tor's cell decryption. This design minimizes the computational load of the measurer while replicating the cryptographic operations that the target would perform on normal traffic, which is needed to get an accurate estimate of its forwarding capacity. To ensure that the target is correctly decrypting and forwarding cells, the measurer records the contents of each cell sent with probability p (e.g., $p = 10^{-5}$) and checks that the returned content of such cells is correct, reporting failure from the measurement if not.

A measurement slot lasts a constant number of seconds t (see § V). The BWAuth can end the measurement in this slot early due to a failure reported by a measurer. During the measurement slot, the BWAuth receives from the ith measuring process the number of measurement bytes x_j^i that were relayed by the target to the process in the jth second. The BWAuth also receives from the target the number of normal traffic bytes y_j that the target relayed in the jth second. At the end of the measurement, the BWAuth computes per-second sums of measurement traffic: $x_j = \sum_{i=1}^{m} x_j^i$. It limits the per-second normal traffic to the largest value that is consistent with the measurement traffic and the traffic ratio r: $\overline{y}_j = \min(y_j, x_j r/(1-r))$. The BWAuth computes a per-second estimate $z_j = x_j + \overline{y}_j$ of total bytes relayed by the target, and then it sets its capacity estimate to the median: $z = \text{median}(z_1, \ldots, z_t)$. Incorporating the normal traffic results in better capacity estimates, and enforcing the expected ratio limits how much a malicious relay can increase its capacity estimate by reporting more normal traffic than it actually relayed.

The process of performing a relay measurement assumes the measurers' capacities are known. Therefore, when deploying a new FlashFlow team, modifying an existing team, or whenever a measurer's capacity is expected to have changed, the BWAuth must estimate the network forwarding capacity of its measurers. Measuring measurers is easier than measuring relays because (i) only a lower bound on the measurement capacity is needed, as an underestimate will only affect the speed of the measurement process and not its accuracy; and (ii) it is sufficient to measure the speed at which network traffic can be simultaneously sent and received, as the measurer does not relay bytes through Tor. Therefore, to estimate the network forwarding capacity of a measurer, the BWAuth instructs it to use iPerf [3] to exchange bidirectional traffic with each other measurer on the team concurrently. This measurement uses UDP to eliminate the effects of TCP congestion control that are unlikely to affect the measurement of all relays. A 60-second measurement is performed, and the capacity estimate is the median of the per-second speeds reported by iPerf.

C. Measuring a Relay

Measuring a relay potentially involves a sequence of measurements because the measurer capacity required for an accurate measurement is unknown. Instead of using the maximum amount of measurer capacity for each relay, we instead use informed guesses about relays' capacities and allocate only the measurer capacity needed for those guesses. If the measurement indicates that the allocated capacity was sufficient for a given target, then we conclude the measurement process. Otherwise, we perform another measurement of the target with a higher guess and more measurer capacity. This process reduces the total amount of measurer capacity used to measure the entire network.

Measuring Old Relays: When measuring an *old* relay, that is, one that has an existing capacity estimate z_0, we simply use z_0 as a guess for its current capacity. The BWAuth needs to allocate $f \cdot z_0$ total capacity across the measurers, where f is an excess allocation factor. Let c_i denote the network capacity of measurer M_i. The BWAuth can allocate to this measurement any amount a_i of the capacity of M_i subject to $0 \leq a_i \leq c_i$ and $\sum_i a_i = f \cdot z_0$. Optimizing the distribution of measurer capacity to maximize the number of relays that can be measured at once is computationally hard, so we use a greedy heuristic to allocate capacity by repeatedly assigning the measurer with the most residual capacity to use all its remaining capacity or as much as is needed to reach $f \cdot z_0$.

The allocation factor f is defined so that the measurement has a high probability of being accurate and conclusive. It depends on a multiple g that is just large enough so that, for error parameters $\epsilon_1, \epsilon_2 \geq 0$, if a relay with true capacity x is measured using at least gx measurer capacity, then the capacity estimate z is almost certainly greater than $(1-\epsilon_1)x$ and less than $(1+\epsilon_2)x$. The value for g is determined experimentally. In addition to g, f includes a factor $(1+\epsilon_2)/(1-\epsilon_1)$ to ensure that z cannot result from values $x' > x$ for which the measurement errors may be larger. The excess allocation factor is thus $f = g(1+\epsilon_2)/(1-\epsilon_1)$.

Using the capacity allocations, the team performs a measurement and obtains a capacity value z. This value is taken as the new estimate if it is small enough relative to the total measuring capacity that it could only result from a true relay capacity close to z. Specifically, z is the new capacity estimate if $z < \sum_i a_i(1-\epsilon_1)/g$. When this is true, the true relay capacity x must be greater than $z/(1+\epsilon_2)$ and less than $z/(1-\epsilon_1)$, which implies that the estimate is accurate, i.e., that $z \in ((1-\epsilon_1)x, (1+\epsilon_2)x)$. If the capacity estimate z is not sufficiently small, then the relay must be measured again using a higher total measurer capacity. In this case, we set $z_0 = \max(z, 2z_0)$, which ensures the allocated capacity will at least double, and we repeat the measurement with the updated capacity estimate z_0.

If the original estimate z_0 is the true capacity, then the measurement process will almost certainly conclude after one measurement. This is true because the measuring capacity was chosen to be large enough that $z < (1+\epsilon_2)z_0$ with high probability, and when that is true the condition to use the z as the new capacity estimate is satisfied: $z < z_0(1+\epsilon_2) = \sum_i a_i(1-\epsilon_1)/g$.

Measuring New Relays: When measuring a *new* relay, which has no capacity estimate, we initially guess the capacity based on the capacity distribution of existing relays. New relays either have never been seen before or were last measured so long ago (e.g., a month) that their capacity measurements are no longer reliable estimates. For such relays, we use as a capacity estimate z_0 the 75th percentile measured capacity among Tor relays over the past month. When this value is sufficiently smaller than the maximum capacity measurable, this allows us to devote less measurer capacity to the measurement. We then expect that one measurement will be sufficient for 75% of new relays. Given this estimate, the measurement proceeds as with old relays, where again if the measurement z is too high relative to the allocated capacity, the relay is scheduled for another measurement with estimate $z_0 = \max(z, 2z_0)$.

D. Measuring the Network

To measure all relays in the network, the BWAuths periodically determine the measurement schedule. The schedule determines when and by whom a relay should be measured. We assume that the BWAuths have sufficiently synchronized clocks to facilitate coordinating their schedules. A measurement schedule is created for each measurement period, the length p of which determines how often a relay is measured. We use a measurement period of $p = 24$ hours.

To help avoid active denial-of-service attacks on targeted relays, the measurement schedule is randomized and known only to the BWAuths. Before the next measurement period starts, the BWAuths collectively generate a random seed (e.g., using Tor's secure-randomness protocol [4]). Each BWAuth can then locally determine the shared schedule using pseudorandom bits extracted from that seed. The algorithm to create the schedule considers each measurement period to be divided into a sequence of t-second measurement slots. For each old relay, slots for each BWAuth to measure it are selected uniformly at random without replacement from all slots in the period that have sufficient unallocated measurement capacity to accommodate the measurement. When a new relay appears, it is measured separately by each BWAuth in the first slots with sufficient unallocated capacity. Note that this design ensures that old relays will continue to be measured, with new relays given secondary priority in the order they arrive.

IV. Security Analysis

Properties: FlashFlow is designed to be secure against an adversary that attempts to cause incorrect measurements. For the specific application of load balancing, we are particularly focused on preventing malicious relays from obtaining incorrectly large capacity estimates and honest relays from obtaining incorrectly small estimates. The threat model includes an adversary that runs malicious relays, malicious clients, some malicious BWAuths, and some malicious DirAuths. Honest BWAuths are assumed to use honest measurement teams. We require that a majority of BWAuths are honest. Our threat model does not include an AS-level adversary, one that runs a Sybil attack, or one that can change capacities quickly.

The FlashFlow design requires a target relay to demonstrate its capacity in a way that cannot be falsified. Thus, rather than depending on self-reports (as TorFlow does fundamentally), FlashFlow has measurers actually send and receive the same cells as normal Tor clients would. Moreover, the sent cell contents are randomly generated and the received contents checked at random to ensure that the target is properly receiving, decrypting, and returning the cells during the measurement. A relay that forges responses (e.g., to skip decryption or to send early before receiving) is detected with overwhelming probability when a response cell is checked due to the random contents, and a response cell is checked with probability p. As a result, a malicious relay that forges k responses has approximately a $(1-p)^{-k}$ chance of evading detection.

Relays do self-report normal client traffic during measurement. However, that client traffic is supposed to be limited to at most a fraction r of the total traffic, and during aggregation the BWAuth limits the reported normal traffic to be at most r times the total. A malicious relay could send no normal traffic but report the full amount, and it could thereby inflate its capacity estimate by at most a factor of $1/(1-r)$.

Several features also prevent a relay from providing high capacity only while it is being measured. Measurement by any given BWAuth is performed at a randomly selected slot in a measurement period, and the randomness is known only to the BWAuths. Furthermore, the relay is measured by multiple BWAuths at separate random times, and the median of the estimates is used. For an adversary that does not control a BWAuth, an attempt to provide high capacity only during a fraction $q < 1/2$ of measurement slots will fail with probability at least 0.5. More accurately, with n BWAuths the probability is $\sum_{k=n/2}^{n} Pr[B(n, 1-q) = k]$, with $B(n,p)$ binomially distributed. Relays are notified of a measurement at its beginning, but due to the shortness of the measurement slot (e.g. $t = 30$ seconds), a malicious relay has little time to adjust its capacity dynamically. The frequency with which relays are measured also forces malicious relays to be able to consistently support their measured capacities. A relay is measured once every period by each BWAuth, so even after a relay has been measured by a majority of BWAuths (which is expected to take a majority of the period), it can only reduce its capacity until the next period. The efficiency of FlashFlow allows the measurement period to be relatively short (e.g. every 24 hours) and thus gives little time for a malicious relay to act at a reduced capacity.

Another security benefit of a randomized measurement schedule is that it limits the opportunity for malicious clients to perform a targeted denial-of-service attack. An adversary may try to do this in order to reduce the measured capacity of certain honest relays, which would cause Tor's load balancing to shift traffic away from them. However, assuming the adversary controls no BWAuth, the adversary cannot predict when an honest relay will be measured and must perform any denial-of-service attack during most of the measurement slots in order to expect to affect the median measurement. The adversary may try to detect when a relay has started being measured, but we argue this would lead to too many false positives to be practical: (1) the attacker has very little time ($<$ 15 seconds given a 30 second measurement) to decide the target relay is being measured as it will then need to DoS the relay for the majority of the measurement in order to affect the *median* per-second throughput during the measurement, (2) for relays utilized on average less than what r would allow during a measurement, the attacker will never notice a drop in throughput, and (3) for relays that regularly dip below what r would allow, it would look like the relay is being measured many times per day.

Finally, we observe that it is difficult for an adversary to prevent relays from being measured by flooding the network with new relays. Old relays are guaranteed to be measured during a measurement period because they are scheduled first. New relays are given second priority, and moreover they are served on a first-come, first-served basis, so benign new relays are eventually measured.

Limitations: In some cases malicious relays may be able to cause FlashFlow to obtain larger capacity estimates than the relays could sustain in Tor. We argue that these limitations are shared by Tor's existing system, TorFlow, and that FlashFlow's security and accuracy advantages make it a significant improvement. Moreover, we suggest ways to improve FlashFlow in the future to mitigate these issues.

One limitation is that an adversary that has access to multiple IP addresses on the same machine can surreptitiously run multiple relays on the machine simultaneously. Tor only accepts two relays at the same IP address (a restriction that was instituted as a defense against falsely obtaining a large total bandwidth weight [11]). FlashFlow is likely to measure multiple relays on the same machine at separate times, so each relay would obtain a capacity estimate that is the capacity of the shared machine. Tor considers this a Sybil attack, and it currently requests that each relay operator identifies all relays that they run with the MyFamily option [8]. Moreover, Tor has made use of systems designed to detect Sybils on its network [37]. Pairs of MyFamily relays (or suspected Sybils) can be measured simultaneously with FlashFlow to determine if they share the same Tor capacity, and then the measured capacity averaged over the members of a connected set. The current TorFlow system shares this issue, as the speed measurements are performed at different times, and an adversary can detect when one of its relays is being measured and reserve all capacity for the measurement circuit [23, 34].

Another limitation is that FlashFlow measurements are so short that they might measure the *burst* speed of a host rather than its sustainable Tor capacity. For some ISPs and hosting providers, higher burst capacities are supported than are consistently achievable. This can be true as a matter of practice, as a network shared by many hosts may occasionally be underutilized, or as a matter of policy, as providers may institute price-based limits on the speed of network traffic from a host. In the former case, if the burst speed is due to variable congestion of shared resources, then we expect the median of the separate and randomly scheduled measurements by different BWAuths to produce good estimates of average performance. In the latter case, if such limits are applied faster than half the length of our measurement slots (e.g., in less than 15 seconds), then FlashFlow should obtain a sustainable capacity estimate. Moreover, we again observe that this issue currently affects TorFlow, which performs relatively short downloads of files (none larger than 64 MiB). An adversarial relay that can change its capacity quickly can do so such that it is measured at a high capacity that it otherwise never provides to clients; periodic audits or network-wide accounting (à la PeerFlow [23]) could help mitigate this attack.

We further note that FlashFlow is designed to measure Tor capacity and not to detect if client traffic is actually relayed. A malicious relay can send little to no real client traffic while obtaining accurate capacity estimates from FlashFlow by only sending traffic on measurement circuits. This is an additional limitation shared with TorFlow, in which the measurement circuits are easily detected [34] and thus weights can be obtained while denying all traffic not used for measurement [23]. Such behavior seems highly observable, however, and so we leave detecting such misbehavior as a future enhancement.

V. Network Experiments

We measure and evaluate FlashFlow's performance and accuracy with a set of network experiments.

A. Preliminary Setup and Analysis

Internet Vantage Points: To perform realistic measurements on the Internet, we obtain hosts from a set of geographically diverse network locations. Table I summarizes the characteristics of our hosts located in Fremont, CA (US-SW), Santa Rosa, CA (US-NW), Washington, DC (US-E), Bangalore, India (IN), and Amsterdam, Netherlands (NL). Our host choices are motivated by the need to demonstrate FlashFlow obtains accurate results under a variety of network conditions (e.g. latency and packet loss) at the measurers and relay.

Because network bandwidth is an important factor that will affect our experiments, and because the supported bandwidth was not advertised for all hosts, we empirically estimate it using iPerf [3] (a network performance measurement tool). We perform a set of experiments where for each host we instruct all other hosts to perform a UDP iPerf measurement to it at the same time for 60 seconds. We sum together the per-second

TABLE I: Summary of the hosts used in Internet experiments

	US-SW	US-NW	US-E	IN	NL
Virtual	No	Yes	No	Yes	Yes
Network Type*	D.C.	D.C.	Res.	D.C.	D.C.
BW (claimed) (Mbit/s)	1000	1000	1000	N/A	N/A
BW (measured) (Mbit/s)	954	946	941	1076	1611
RTT to US-SW (ms)	0	40	62	210	137
CPU cores	8	8	12	2	2
RAM (GiB)	32	4	32	4	4

* Network type is datacenter (D.C.) or residential (Res.)

results from each host and present the median of the summed per-second results in the "BW (measured)" row in Table I. All three of the US hosts are clearly limited to about 1 Gbit/s. IN and NL achieve higher throughput despite their hosting provider making no claims about their capacity.

Tor Processing Limits: We evaluate Tor's processing limits to estimate the throughput that a FlashFlow team must support in order to measure the fastest Tor relays. We set up a lab experiment that attempts to maximize throughput while minimizing the effect of limiting factors including network latency, congestion and flow control algorithms in TCP and in Tor, the capacity of the underlying network, and the number of Tor circuits and TCP sockets used during the measurement. Over a 120-second measurement, we found that the maximum capacity of a Tor relay was 1.25 Gbit/s, achieved while using 20 TCP sockets. Tor reached 100% CPU utilization during this measurement, indicating that Tor's single-threaded scheduling is the performance bottleneck. This measurement result constitutes an upper bound of the capacities that FlashFlow should be able to measure, and it is consistent with the fastest claimed capacity of a Tor relay, which was 998 Mbit/s in July 2019 [7].

Because we will use our US-SW host to run target relays in our Internet experiments, we also establish ground-truth Tor capacity on it by running an experiment similar to the one described above. We run a relay on US-SW, and use the remaining machines to run Tor processes that support the measurement of US-SW. The target relay on US-SW achieves a maximum median throughput of 890 Mbit/s while consuming 95–100% of a CPU core, and so we will use that amount as its ground-truth capacity.

FlashFlow Implementation and Setup: We implement FlashFlow as a 1,200-line patch to Tor v0.3.5.7 containing measurer- and relay-side measurement support and a 1,300-line C/Rust program that controls FlashFlow measurers. The experimental setup for the remainder of this section is as follows. US-SW runs a single target Tor relay. Some combination of the remaining hosts (US-NW, US-E, IN, and NL) measure the target relay. We configure FlashFlow with the following settings, which were determined through a sequence of experiments (details are excluded for space reasons): the number of measurement sockets $s = 160$ (the s that maximizes throughput on the slowest host); the multiplier $g = 2.25$ (the smallest g that yields sufficient accuracy); the measurement duration and strategy is to take the median throughput achieved in $t = 30$ seconds (reasonable balance between time-to-result and accuracy); and error bounds of $\epsilon_1 = 0.20$ and $\epsilon_2 = 0.05$.

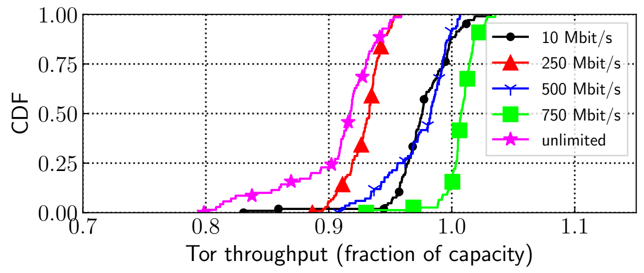


Fig. 2: Evaluation of FlashFlow's accuracy from 24 hours worth of 30 second experiments with multiplier $g = 2.25$. CDFs are over the median per-second throughput measured by each team.

B. Measurement Accuracy

We evaluate FlashFlow's accuracy with and without client background traffic.

Without Client Background Traffic: We conduct a set of Internet experiments in which we configure a target relay on US-SW and form measurement teams from all possible unique subsets of the remaining machines from Table I. We set throughput limits of 10, 250, 500, 750, and unlimited Mbit/s on the target; for each such limit we test how well all measurement teams can measure it, where each measurer in each team is limited to its share of the factor f of measurer capacity that is necessary to measure the target using $g = 2.25$. Each such measurement (a team measuring a throughput-limited relay) runs for 30 seconds and is repeated 7 times over the course of 24 hours. The result of each measurement is the median per-second throughput over the 30 second period.

Figure 2 shows the accuracy of our measurements, categorized by the Tor capacities at the target. Across all configured capacities, all but one experiment (99.8%) produces results within $\epsilon_1 = 0.20$ and $\epsilon_2 = 0.05$. FlashFlow measures within 11% error (0.89–1.11 times capacity) in 95% of experiments.

With Client Background Traffic:

To evaluate FlashFlow's ability to measure a relay with realistic client background traffic, we run a Tor relay on US-SW and connect it to the real Tor network (for convenience, we use another identical machine in the same datacenter). We run the relay for 60 days before starting any FlashFlow measurements so that it is measured by the existing BWAuths, earns the Guard flag, and attracts a significant amount of client traffic. The relay is configured to limit its Tor throughput to 250 Mbit/s, and we measure it with one FlashFlow measurer running on NL. FlashFlow reports for each second the amounts of measurement traffic and background traffic. We test ratios r between 0.1 and 0.5 to limit background traffic (§ III-B).

Our measurements show client background traffic of 50 Mbit/s on average before and after measurement. During measurement, the total traffic is at the relay's capacity of 250 Mbit/s. When $r \leq 50/250 = 0.2$, background traffic during measurement is consistently measured at $250r$, and for larger r the background traffic stays at its original rate.

We conclude that FlashFlow successfully limits background traffic to at most the configured ratio r. Because Tor's utilization rate has been at about 0.5 on average for the last several years [7], we recommend $r = 0.25$, which should reduce background traffic rates during measurement by about half while limiting a malicious relay's measurement inflation factor to $1/(1-r) = 1.33$ (see § IV). We emphasize that the ratio only affects performance when a relay's utilization rate rises above r. Moreover, it is only applied during 30 second measurement that happens once per day per BWAuth, i.e., an insignificant 0.035% of each day per BWAuth. For more details on this experiment, see Appendix A.

VI. Simulation Experiments

Having demonstrated FlashFlow's efficacy in measuring live relays over the Internet in § V, we now use simulation to conduct a more complete analysis of measurement efficiency, cost, accuracy, and performance. Simulation allows us to analyze a full Tor network deployment of FlashFlow, which is not possible in our limited network deployment in § V.

A. Measurement Efficiency and Cost

Efficiency: We evaluate the efficiency of FlashFlow in measuring the entire Tor network in terms of its speed. To estimate these values, we simulate measurement of the network by a single team. We use a greedy scheduler to determine the fastest that we can measure the entire network. Then we replay the appearance of new relays in the consensus and determine how efficiently they can be measured as well.

We determine the state of the Tor network over July 2019 using archived Tor consensuses and descriptors [7]. We estimate the capacity of relay r at time t to be the minimum of the rate limits set in the relay's descriptor at t and the largest observed bandwidth for r in the period June–August 2019. Among all relays, the largest capacity thus determined for July 2019 is 998 Mbit/s.

We estimate how fast FlashFlow could measure the entire network for each day in July 2019. For this estimate, we use the first consensus in the day, and we assume that all of the relays in the network have been measured before and thus have capacity estimates. We greedily assign relays to each slot in order, with each assignment choosing the largest relay for which there is available capacity to measure. We use a measurement team consisting of 3 measurers with 1 Gbit/s capacity each. This team has capacity that is just larger than the minimum required to accurately measure the largest relay seen, which due to the excess factor $f = 2.84$ and maximum capacity of 0.998 Gbit/s is 2.84 Gbit/s.

The result for the median day is that 5 hours (i.e. 599 30-second slots) are needed to measure the entire network, with a minimum of 4.9 and a maximum of 5.1. The schedule measures a median of 6,419 relays (min: 6,355, max: 6,528) with a median total capacity of 608 Gbit/s (min: 592, max: 621). This speed suggests that the entire network could be measured at least every 24 hours with significant spare capacity to measure new relays as they join the network.

We next estimate how quickly new relays can be measured. A relay is considered new if it has not been seen in the last month. We consider each consensus in July 2019 and assume relays in the first consensus are not new. During this time, there is a median of 3 new relays in a consensus (min: 0, max: 98). We use as the new-relay capacity estimate the 75th percentile advertised bandwidth from descriptors in June 2019, which is 51 Mbit/s. The simulation result is that the median time to measure new relays in a consensus is 30 seconds (min: 0 minutes, max: 13 minutes). These results show that new relays can be measured within minutes even while FlashFlow re-measures the entire network every 24 hours.

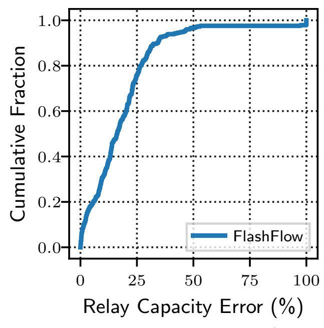
Cost: To calculate the amount of added traffic to the current Tor network per day, we calculate fCt, with the excess factor $f = 2.84$, the total capacity of the network $C = 608$ Gbit/s, and the measurement duration $t = 30$ seconds. This produces 5.89 TiB of traffic per day. During June–August 2019, the Tor network forwarded about 200 Gbit/s, or about 1,965 TiB per day, thus each FlashFlow team adds approximately 0.3% to Tor's current load, which is a reasonably small amount.

Deploying FlashFlow today with our recommended parameters requires similar bandwidth capacity from its operators as does TorFlow. With that capacity, FlashFlow can measure a growing Tor network up to $24/5 = 4.8$ times the current size every day. Moreover, by accepting a lower measurement frequency, the same measurer capacity scales even further. Otherwise, additional measurer capacity is necessary.

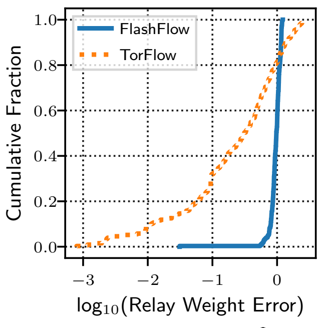
B. Measurement Accuracy and Performance

We evaluate FlashFlow in a full Tor network deployment using Shadow [20], a discrete-event network simulator and a standard tool for conducting Tor performance experiments [31]. We configure a private Tor test network in Shadow that is 5% of the size of the public network and contains: 3 DirAuths; 328 relays; 397 TGen clients that use Tor Markov models to generate the traffic flows of 40k Tor users [22]; and 40 TGen clients that mirror Tor's performance benchmarking process by repeatedly downloading 50 KiB, 1 MiB, and 5 MiB files (timeouts are set to 15, 60, and 120 seconds, respectively). The relays were sampled from Tor's consensus files from January 2019 [7] and placed in the closest city in Shadow's Internet map according to IP address and following best practices [19]. Each relay r is configured with a true capacity C_r that is equal to the maximum observed bandwidth of the corresponding relay in Tor during January 2019.

Accuracy: To measure accuracy, we first run a base FlashFlow simulation (using our implementation and configuration from § V) in which FlashFlow uses 3 measurers with capacities of 1 Gbit/s each to measure the Tor network and produce a bandwidth file containing a capacity estimate $\overline{C}_r$ and weight $\overline{W}_r$ for each relay r. We repeat the simulation with TorFlow, which produces a bandwidth file with weights only (i.e., weights $\overline{W}_r$ are generated directly rather than deriving them from capacity estimates $\overline{C}_r$). We use the capacity estimates $\overline{C}$, weight estimates $\overline{W}$, and the corresponding ground truths (i.e., true capacities C_r and weights $W_r = C_r/\sum_s C_s$)



(a) Relay Capacity Error $\widehat{C}_r$



(b) Relay Weight Error $\widehat{W}_r$

Fig. 3: Relay measurement error during concurrent relay measurement in Shadow simulations. The corresponding network capacity error $\widehat{C}$ is 14% for FlashFlow, while the corresponding network weight error $\widehat{W}$ is 4% for FlashFlow and 29% for TorFlow (see Table III).

to compute relay capacity error $\widehat{C}_r = |1 - (\overline{C}_r/C_r)|$, relay weight error $\widehat{W}_r = \overline{W}_r/W_r$, network capacity error $\widehat{C} = 1 - (\sum_r \overline{C}_r / \sum_r C_r)$, and network weight error $\widehat{W} = \sum_r |\overline{W}_r - W_r|/2$. See Appendix B for a summary of these variables.

Figure 3 shows the relay capacity and weight error as CDFs over all relays. Figure 3a shows that both the median and inter-quartile range of capacity error *across relays* is 16%, and the corresponding network capacity error $\widehat{C}$ is only 14% *in total*. Figure 3b compares the relay weight error for FlashFlow and TorFlow; $x = 0$ represents ideal relay weighting, and each unit on the x-axis represents a 10× increase in error. We observe that more than 80% of relays are underweighted by TorFlow compared to their ground truth capacity. FlashFlow shows considerable improvement in relay weighting, with a total of only 4% network weight error $\widehat{W}$ compared to 29% for TorFlow. We expect that the significant reduction in weight errors that we observed for FlashFlow will result in both better load balancing and better performance for Tor users.

Performance: To measure performance, we use the bandwidth files produced by FlashFlow and TorFlow in the above simulations to run 3 new simulations for each system; one simulation is configured with normal (100%) traffic load, one with 15% extra (115%) traffic load, and one with 30% extra (130%) traffic load. In all simulations, Tor is configured to form a consensus with the previously measured relay weights, thus client load is balanced according to these weights.

Figures 4 and 5 show considerable improvement in performance when using the FlashFlow weights compared to TorFlow across *all metrics and benchmarks*. Overall, our simulations demonstrate that FlashFlow is significantly more capable of balancing load in Tor than is TorFlow.

Figure 4 shows that the FlashFlow benchmark clients *outperform* the TorFlow benchmark clients across all transfer sizes: in the 100%-loaded simulations, the median of 50 KiB, 1 MiB, and 5 MiB transfer times decreases by 15%, 29%, and 37%, respectively. FlashFlow also yields more *consistent* client performance: in the 100%-loaded simulations, the standard deviation of 50 KiB, 1 MiB, and 5 MiB transfer times decreases by 55%, 61%, and 41%, respectively. We also observe that FlashFlow better supports *network growth* because the performance improvements increase as the network becomes more loaded: relative to TorFlow, the median 1 MiB transfer time in FlashFlow decreases by an additional 28% and 29% when the network is 15% and 30% more loaded, respectively. Surprisingly, performance in the 130%-loaded FlashFlow simulation was still better than performance in the 100%-loaded TorFlow simulation, across all transfer sizes.

Figure 5a shows that the median rate of transfer timeouts decreases by 100% in all FlashFlow simulations, compared to median transfer failure rates of 5%, 10%, and 23% for TorFlow in the 100%-, 115%-, and 130%-loaded simulations, respectively. Almost no timeouts are recorded by clients when they build circuits using the FlashFlow weights, indicating that load in the network is much more balanced and that network bottlenecks that were present under the TorFlow weights have been reduced or eliminated. Fewer timeouts for Tor clients will result in a less frustrating user experience.

Finally, Figure 5b provides further evidence that the FlashFlow weights result in a more balanced network capable of handling additional traffic load. Increasing client-traffic load by 15% and 30% results in a 15% and 29% increase in the median Tor throughput (summed over all relays) in FlashFlow as expected, indicating that the network is able to process the additional load without a significant increase in network congestion. However, increasing client-traffic load by 15% and 30% results in only a 12% and 18% increase in TorFlow, respectively, indicating that the additional load and resulting increase in network congestion causes a corresponding reduction in attainable inter-relay TCP connection throughput.

VII. Related Work

Load Balancing in Tor: Several systems for load balancing in Tor have been proposed. Load-balancing systems produce the relay weights that clients use to select paths, and in some cases the relay capacities can also be determined. A comparison of these systems appears in Table II. It shows the server bandwidth as currently deployed (for TorFlow) or assumed to be deployed (for the rest), the demonstrated success factor of a weight-inflation attack, if the system provides capacity values in addition to weights for load balancing, and how long it takes to produce weights for the entire network. We observe that for some increase in required server bandwidth, FlashFlow provides increased security and speed, and it can be used for capacity estimates as well as load balancing.

The Tor network uses TorFlow [28] to estimate relays' capacities and assign weights accordingly. We discuss TorFlow in § II. TorFlow is vulnerable to attacks [10, 23, 34], the most straightforward of which is that a malicious relay can falsely report very high bandwidth information in its descriptor [10], increasing its final weight regardless of its performance measurements. Such attacks have been demonstrated to increase the weight of a Tor relay by 89× [34] to 177× [23]. Data

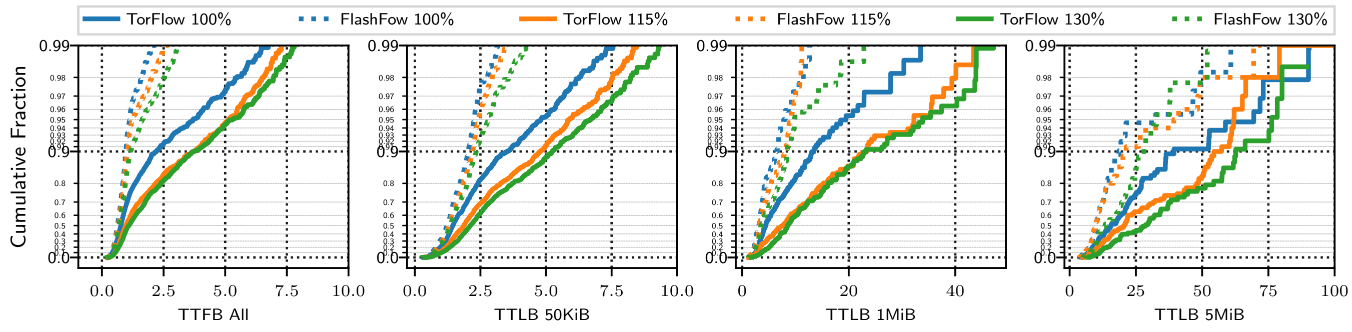
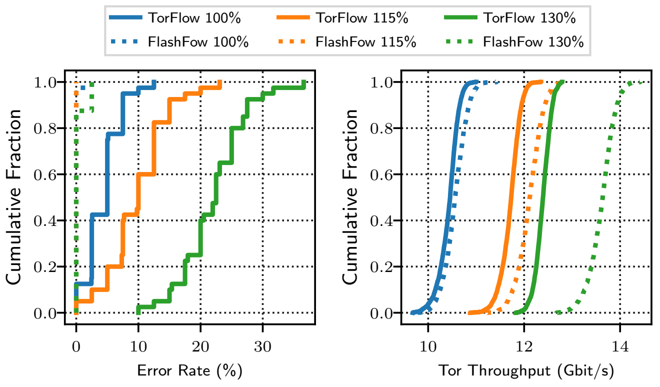


Fig. 4: Performance results when using TorFlow and FlashFlow weights in Shadow simulations with normal (100%) and extra (115%, 130%) traffic load. Shown is the CDF over the time to first and last byte of 50 KiB, 1 MiB, and 5 MiB transfers by performance benchmark clients. The CDFs are plotted with a logarithmic y-axes to highlight the long-tail performance as a measure of the impact on usability. Our results demonstrate that FlashFlow outperforms TorFlow at all load levels due to more accurate load balancing.



(a) Benchmark Errors **(b)** Tor Network Throughput

Fig. 5: Performance results when using TorFlow and FlashFlow weights in Shadow simulations with normal (100%) and extra (115%, 130%) traffic load. (a) The fraction of benchmark client transfers that failed (timed out) is usually 0 for FlashFlow but increases with the extra traffic load in TorFlow. (b) The Tor network throughput (for every second, sum of relays' Tor throughput) increases roughly linearly with the traffic load for FlashFlow, but reaches a ceiling in TorFlow due to high utilization of links that become bottlenecks due to improperly balanced load.

TABLE II: Comparison of Tor load-balancing systems

	Server BW	Attack Advantage	Capacity Values*	Speed
TorFlow†	1 Gbit/s	177×	◐	2 days
EigenSpeed	0‡	21.5×$^\diamond$	○	1 day
PeerFlow	0‡	10×$^\diamond$	◐	14 days$^+$
FlashFlow	3 Gbit/s	1.33×	●	5 hours

* Values provided (●), can be inferred (◐), or unavailable (○).
† Attributes apply also to SmarTor and Simple Bandwidth Scanner.
‡ Relays measure each other using existing client traffic.
$^\diamond$ With 20% trusted relays (by number or weight).
$^+$ Time to measure largest 96.8% of relays.

from TorFlow's BWAuths [2, 30] indicate that a single 1 Gbps scanner takes at least 2 days to measure the entire network.

SmarTor [9] decentralizes the operation of the BWAuths using a blockchain and trusted execution environments. Like TorFlow, it measures relay capacity by downloading a file through the relay. It thus remains vulnerable to bandwidth-inflation attacks demonstrated against TorFlow. We do not include SmarTor in Table II because its contributions over TorFlow are not to the measurement technique itself. Consequently, its measurement attributes can be assumed to be similar to that of TorFlow. Simple Bandwidth Scanner [25] is a more easily maintained replacement for TorFlow that produces the same results as TorFlow and does not address any of its fundamental weaknesses, thus we do not include it either.

EigenSpeed [32] uses a peer-measurement approach in which every relay records the average per-stream throughput with every other relay and reports this vector to the Tor DirAuths. The DirAuths combine the vectors into a matrix and compute the eigenvector of that matrix as the relay weights. This computation is done iteratively and is initialized with the weights of trusted relays. During and after the computation, relays can be marked as malicious due to atypical changes in or unusual final values of their weights, and these relays are effectively removed from the network. EigenSpeed observations are per-flow throughputs rather than total relay capacity. EigenSpeed is vulnerable to several attacks [23], including a Sybil attack, an "increase framing attack", and an "targeted liar attack". In the last attack, malicious relays can inflate their total weight to 7.4–28.1 times the weight they deserve, depending on the number of trusted relays.

In PeerFlow [23], relays periodically report to the DirAuths the total number of bytes they exchange with each other. The DirAuths then securely aggregate the traffic data to produce relay weights. In the process of determining weights, PeerFlow produces lower bounds on relay capacities that can be used as capacity estimates. PeerFlow requires a fraction τ of relay weight that is trusted, and the adversary can obtain weights for his relays inflated by a factor of $2/\tau$. If $\tau = 0$, then a sufficiently large adversary (i.e. relative weight above 4%) can eventually get an arbitrarily large relative weight. PeerFlow also limits how quickly a malicious relay's weight can increase from one measurement period to the next. Based on the suggested parameters, a malicious relay can inflate its claimed capacity by a factor of 4.5 (see [23, Theorem 1]).

Compared to these systems, FlashFlow has much better protection against weight inflation in both the short and long term as it has an inflation factor of 1.33 at all times. It also allows the entire network to be measured in hours rather than days. FlashFlow requires higher measurement-server bandwidth than other systems, but it is still not high (3 Gbit/s), especially compared to total Tor network capacity (>400 Gbit/s).

We note some additional systems superseded by later work or that do not directly produce load-balancing weights. Snader and Borisov [33] propose a simple form of EigenSpeed's peer measurement that takes the median of pairwise speed observations. It uses an unweighted median and is thus vulnerable to a Sybil attack. TightRope [13] assumes that capacity weights already exist for the relays and then considers how to choose paths to optimally balance load. Using differential privacy, the current load on all relays is shared with a server that computes a distribution for clients to use when building new circuits. Wang et al. [36] propose Tor clients use lightweight active measurements that use latency as an indicator for congestion, detect congested relays, and automatically avoid using them.

Other Related Work: Speed tests such as Ookla [6] are primarily intended for home users to test the throughput of their devices, wireless router, or their ISP's connection. iPerf [3] can achieve high throughput at the transport layer over both UDP and TCP. Prasad et al. [29] describe bandwidth-estimation techniques, focusing on efficient techniques such as packet pairs and trains. Feamster and Livingood discuss the challenges of Internet throughput measurement even when allowing the measurement to fully utilize bandwidth [17].

VIII. Conclusion

Tor's load-balancing system is insecure, its capacity estimation is biased, and its weights have uncontrollable variance. To solve these problems, we present FlashFlow, which actively measures Tor relays with limited effect on normal traffic. We implement FlashFlow, conduct experiments, and show it accurately, securely, and quickly measures the Tor relay capacities. We also show that these capacities improve the load-balancing of Tor. For a discussion of possible future work, see Appendix C.

Acknowledgments

This work was supported by the Office of Naval Research.

References

[1] "More consensus weight problems," https://lists.torproject.org/pipermail/tor-relays/2015-June/007167.html, June 2015.

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

[3] "iPerf - The ultimate speed test tool for TCP, UDP, and SCTP," https://iperf.fr/, December 2018.

[4] "Tor Shared Random Subsystem Specification," https://github.com/torproject/torspec/blob/master/srv-spec.txt, July 2018.

[5] "What's wrong with my relay? (Consensus Weight Jumping Around)," https://tor.stackexchange.com/questions/16747/, February 2018.

[6] "Speedtest by Ookla - The Global Broadband Speed Test," https://www.speedtest.net/, September 2019.

[7] "Tor Metrics," https://metrics.torproject.org, August 2019.

[8] "Tor Project: manual," https://2019.www.torproject.org/docs/tor-manual.html.en, August 2019.

[9] G. Andre, D. Alexandra, and K. Samuel, "Smartor: Smarter tor with smart contracts: Improving resilience of topology distribution in the tor network," in *ACSAC*, 2018.

[10] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-resource Routing Attacks Against Tor," in *WPES*, 2007.

[11] K. Bauer and D. McCoy, "No more than one server per IP address," Tor Proposal 109, March 2007.

[12] A. Biryukov, I. Pustogarov, and R.-P. Weinmann, "Trawling for tor hidden services: Detection, measurement, deanonymization," in *IEEE S&P*, May 2013.

[13] H. Darir, H. Sibai, N. Borisov, G. Dullerud, and S. Mitra, "TightRope: Towards Optimal Load-balancing of Paths in Anonymous Networks," in *WPES*, 2018.

[14] R. Dingledine, "The lifecycle of a new relay," https://blog.torproject.org/lifecycle-new-relay, September 2013.

[15] R. Dingledine and N. Mathewson, "Tor Protocol Specification," https://gitweb.torproject.org/torspec.git/, November 2018.

[16] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," in *USENIX Security*, 2004.

[17] N. Feamster and J. Livingood, "Internet Speed Measurement: Current Challenges and Future Recommendations," https://arxiv.org/abs/1905.02334, September 2019.

[18] D. Goulet, "kist: Poor performance with a small amount of sockets," Tor Trac Ticket 29427, 2018.

[19] R. Jansen, K. S. Bauer, N. Hopper, and R. Dingledine, "Methodically Modeling the Tor Network," in *CSET*, 2012.

[20] R. Jansen and N. Hopper, "Shadow: Running Tor in a Box for Accurate and Efficient Experimentation," in *NDSS*, 2012.

[21] R. Jansen, M. Traudt, J. Geddes, C. Wacek, M. Sherr, and P. Syverson, "KIST: Kernel-Informed Socket Transport for Tor," *ACM TOPS*, vol. 22, no. 1, December 2018.

[22] R. Jansen, M. Traudt, and N. Hopper, "Privacy-Preserving Dynamic Learning of Tor Network Traffic," in *ACM CCS*, 2018.

[23] A. Johnson, R. Jansen, N. Hopper, A. Segal, and P. Syverson, "PeerFlow: Secure Load Balancing in Tor," *PoPETs*, 2017.

[24] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users Get Routed: Traffic Correlation on Tor By Realistic Adversaries," in *ACM CCS*, 2013.

[25] Juga, "How Bandwidth Scanners Monitor The Tor Network," https://blog.torproject.org/how-bandwidth-scanners-monitor-tor-network, April 2019.

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

[27] M. Nasr, A. Bahramali, and A. Houmansadr, "DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning," in *ACM CCS*, 2018.

[28] M. Perry, "TorFlow: Tor Network Analysis," in *HotPETs*, 2009.

[29] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy, "Bandwidth estimation: metrics, measurement techniques, and tools," *IEEE Network*, vol. 17, no. 6, 2003.

[30] T. Ritter, "The longer a bwauth runs, the slower it goes ," Tor Trac Ticket 17482, 2015.

[31] F. Shirazi, M. Goehring, and C. Diaz, "Tor Experimentation Tools," in *IWPE*, 2015.

[32] R. Snader and N. Borisov, "EigenSpeed: Secure Peer-to-Peer Bandwidth Evaluation," in *IPTPS*, 2009.

[33] ——, "Improving security and performance in the tor network through tunable path selection," *IEEE TDSC*, 7 2011.

[34] F. Thill, "Hidden Service Tracking Detection and Bandwidth Cheating in Tor Anonymity Network," Master's thesis, Univ. Luxembourg, 2014.

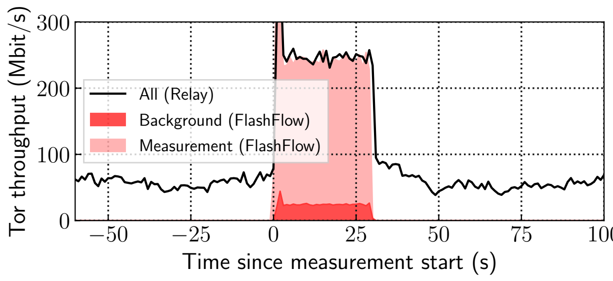


Fig. 6: Tor throughput during measurements of a relay with $r = 0.1$ and client background traffic as reported by the FlashFlow and by the relay. The shaded regions are stacked, and FlashFlow reports their median per-second sum as its result.

TABLE III: Capacity and Weight Error Analysis

Symbol	Definition	Note	Description
C_r	configured		true capacity for relay r
$\overline{C}_r$	measured	†	estimated capacity for relay r
$\widehat{C}_r$	$\lvert 1 - (\overline{C}_r / C_r) \rvert$	†	capacity error for relay r
$\widehat{C}$	$1 - (\sum_r \overline{C}_r / \sum_r C_r)$	†	network capacity error
W_r	$C_r / \sum_s C_s$		ideal weight for relay r
$\overline{W}_r$	$\overline{C}_r / \sum_s \overline{C}_s$	‡	estimated weight for relay r
$\widehat{W}_r$	$\overline{W}_r / W_r$		weight error for relay r
$\widehat{W}$	$\frac{1}{2} \sum_r \lvert \overline{W}_r - W_r \rvert$		network weight error

† Undefined for TorFlow since it does not measure capacity.
‡ Defined as "measured" for TorFlow (i.e., it is not derived from $\overline{C}_r$).

[35] M. Traudt, A. Johnson, R. Jansen, and M. Perry, "FlashFlow: A Secure Speed Test for Tor (Parent Proposal)," Tor Proposal 316, April 2020.

[36] T. Wang, K. Bauer, C. Forero, and I. Goldberg, "Congestion-Aware Path Selection for Tor," in *FC*, 2012.

[37] P. Winter, R. Ensafi, K. Loesing, and N. Feamster, "Identifying and characterizing sybils in the tor network," in *USENIX Security*, 2016.

Appendix A
Client Background Traffic Experiment

As described in § V-B, we run an experiment to evaluate the effect of FlashFlow on background client traffic. We collect data about the traffic relayed before, during, and after a FlashFlow measurement. We measure three values for each second of the experiment: the total amount of forwarded traffic as reported by the target relay (through bandwidth events from its Tor control port), the amount of FlashFlow measurement traffic forwarded by the relay as reported by the FlashFlow measurer, and the amount of background traffic at the relay as reported by it to the FlashFlow measurer.

Figure 6 shows the experiment results for a ratio of $r = 0.1$. We tested values of r from 0.1 to 0.5, and the other results were qualitatively similar. In the results, the sum of the background and measurement traffic reported by FlashFlow is indeed equal to the total traffic reported by the relay, as expected. Following the measurements, the relay's throughput immediately returns to the level it was before, demonstrating that FlashFlow has no lingering effect on background traffic levels. We also observe in the figure that background traffic is limited to 25 Mbit/s, which is as expected given the total capacity of 250 Mbit/s and $r = 0.1$. Note that the spike at the beginning of the measurement is due to the Tor relay allowing a one second burst before limiting its own throughput to 250 Mbit/s.

Appendix B
Error Analysis Variables

Table III summarizes the variables used in § VI-B to compare the accuracy of FlashFlow and TorFlow.

Appendix C
Future Work

Our results show that FlashFlow could be used today to improve Tor's performance and resource estimates. Furthermore, FlashFlow could be used as a secure basis for incorporating additional dynamic performance measurements. Such measurements, such as per-relay network and CPU utilization, could provide information about *available* (rather than total) capacity that may further improve Tor's load balancing. The FlashFlow measurements would be used as a starting weight, and then the weights would only be reduced, depending on the dynamic measurements. FlashFlow would thus securely limit the weight of any relay while allowing for improved performance via adjustments based on insecure dynamic measurements, such as self-measurements.

A challenge for future work is accurate measurement of co-resident relays, that is, multiple relays running on the same host machine. Neither TorFlow nor FlashFlow offers approaches to detect such relays or adjust the corresponding measurement results. Potential options for detecting co-resident relays include a manual configuration option that honest relay operators can set to disclose a family of co-resident relays, and an automatic heuristic detection algorithm based on IP address or ASN. Core Tor itself considers each relay to have a single maximum capacity, an obviously untrue assumption considering the difference in communicating with a relay in the same data center versus on another continent. Because FlashFlow supports running a geographically diverse team of measurers, it handles this reality better than TorFlow, but future work could include more intelligent weighting that includes additional non-node properties.