Privacy-Preserving Dynamic Learning of Tor Network Traffic

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

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

Nicholas Hopper
University of Minnesota
hopper@cs.umn.edu

ABSTRACT

Experimentation tools facilitate exploration of Tor performance and security research problems and allow researchers to safely and privately conduct Tor experiments without risking harm to real Tor users. However, researchers using these tools configure them to generate network traffic based on simplifying assumptions and outdated measurements and without understanding the efficacy of their configuration choices. In this work, we design a novel technique for dynamically learning Tor network traffic models using hidden Markov modeling and privacy-preserving measurement techniques. We conduct a safe but detailed measurement study of Tor using 17 relays (~2% of Tor bandwidth) over the course of 6 months, measuring general statistics and models that can be used to generate a sequence of streams and packets. We show how our measurement results and traffic models can be used to generate traffic flows in private Tor networks and how our models are more realistic than standard and alternative network traffic generation methods.

ACM Reference Format:
Rob Jansen, Matthew Traudt, and Nicholas Hopper. 2018. Privacy-Preserving Dynamic Learning of Tor Network Traffic. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3243734.3243815

1 INTRODUCTION

Tor [11] is the most popular anonymous communication system ever deployed. As of May 2018, the Tor network consists of over six thousand volunteer relays that collectively forward over 100 Gbit of traffic per second from over two million daily users [37]. Tor helps its diverse set of users, ranging from regular Internet users to journalists and digital activists to law enforcement and governments, stay safe while browsing the web and communicating online. To achieve unlinkability of communication, Tor clients multiplex all application *streams* over long-lived *circuits* of three independently chosen and globally distributed *relays*. No single point on this path can determine *both* the communication source *and* destination.

Tor is also an open-source research and development platform: researchers can analyze Tor's design and can experiment with novel anonymous communication algorithms and protocols by changing the source code. Experiments with Tor modifications, however, should be conducted without harming the privacy or anonymity of Tor users.[1] Although this safety requirement had originally limited the applicability of research results to the real world, Tor experimentation tools have been developed [6, 24] and have become a popular means [46] of better understanding how changes to Tor's path selection [1, 5, 10, 33, 34, 43, 45, 51, 53] load balancing [18, 21, 25, 27, 32, 40] traffic admission control [3, 17, 19, 23, 29, 36, 54] congestion control [2, 16] and denial of service mechanisms [9, 20, 31, 44] affect Tor performance and security [4].

While Tor experimentation tools have significantly enhanced the state of Tor research, the relevance of the results they produce depends on the accuracy of the network traffic models used during experimentation. Previous models of Tor clients have generally been limited to single file "web" (320 KiB) and "bulk" (5 MiB) downloaders based on relatively dated measurement studies of ethical concern [8, 39]. Although there has been some limited effort to compare network characteristics in experimental Tor networks to reality [22, 23, 51], these comparisons have completely disregarded potentially important Tor network characteristics such as the number of circuits and their distribution across relays and clients, the number of streams and their distribution across circuits, and the distribution of download sizes across streams. As a result, the efficacy of synthetic Tor network traffic models remains largely unknown.

In this paper, we make four major and significant primary contributions that will have immediate impact on Tor security and performance research.

Measuring Tor. First, we conduct a large and detailed measurement study of Tor. We use a recent privacy-preserving measurement tool called PrivCount [26] and 17 relays representing roughly two percent of Tor's bandwidth capacity [37] to repeatedly measure various Tor client and network characteristics over 3 months. Our study provides a deep analysis of Tor traffic, including the number of active and inactive clients, the number of active and inactive circuits and their distribution per client, the number and types of streams and their distribution per circuit, and the number of inbound and outbound bytes and their distribution per stream. We expect these measurements to be useful not only for producing accurate Tor traffic models, but also for gaining a more thorough general understanding of Tor and its usage.

Learning Tor Traffic. Second, we design novel techniques for dynamically but safely learning Tor traffic models using hidden Markov modeling (HMM). We extend the PrivCount measurement tool [26] to support our techniques and use it on our relays to iteratively measure (i.e., train) Tor packet and stream models over a period of 3 months. We evaluate our models and find that, although Tor traffic is highly variable over short timescales, our best model instances fit Tor traffic reasonably well. Our models can be used to generate streams on a circuit and packets on a stream using standard probability distribution generation techniques.

[1] https://research.torproject.org/safetyboard.html

CCS '18, October 15–19, 2018, Toronto, ON, Canada

ACM ISBN 978-1-4503-5693-0/18/10…$15.00
https://doi.org/10.1145/3243734.3243815

Building Traffic Models. Third, we design and develop a set of modeling semantics and a traffic generation tool called TGen that can be used to create arbitrarily complex patterns of behavior; TGen enables configurable control over the creation of TCP connections and the size, schedule, and duration of packet streams. We describe two new client models: one based on the most common protocols used on Tor (i.e., HTTP and BitTorrent [8, 26, 39]), and one that uses our Tor measurement results and HMM stream and packet models as the basis for traffic generation. Our models are relevant across a range of Tor experimentation tools and research topics.

Evaluating Traffic Models. Fourth, we conduct the first known evaluation of the accuracy of Tor traffic models using a private Tor network deployment of 2,000 relays and up to 60,000 clients in Shadow [24]. First, we update Shadow's outdated Internet model, collecting 1.6 million latency measurements using 1,813 globally distributed vantage points and using them to create a Shadow latency model based on city-pairs. Then, we evaluate multiple traffic models in Shadow: we configure Tor to export PrivCount events, tally the events using a local PrivCount deployment, and compare the results to our ground truth measurements of Tor traffic. We find that the Tor client model that utilizes our traffic models yields a network whose characteristics are closest to those we measured in Tor: the single file model is inaccurate in the distribution of bytes per stream, the protocol (i.e., HTTP and BitTorrent) model is inaccurate in the distribution of streams per circuit, and both models produce fewer clients, circuits, and bytes than Tor and our HMM-based model.

Impact. Our Tor measurements and improved traffic models facilitate the meaningful exploration of open Tor research problems. First, our results will improve research conducted using general-purpose packet-level Tor experimentation tools [6, 24]. For example, the evaluation of the efficacy of proposed Tor load balancing algorithms [43] will be more meaningful when the background traffic (e.g., the number and distribution of circuits, streams, and packets) in a Tor test network is more realistic (i.e., more similar to conditions in the public Tor network). Second, our results will improve research that utilizes higher-level Tor flow or circuit simulations that run over long simulated timescales. For example, proposed secure bandwidth measurement algorithms [32, 48] have benefited from experiments that involve many iterations of full network measurement in order to measure feedback effects and the time to reach steady state. These simulations can use our models as generators to ensure an accurate distribution of flows over arbitrary timescales.

The changes to PrivCount and Shadow that were necessary to carry out this research (to measure and make use of our models) have been contributed to the open-source community and have been merged into the PrivCount[2] and Shadow[3] projects. Furthermore, we have released our PrivCount measurement data, TGen HMM models, and Shadow experimentation models (including our updated Internet latency model as well as our private Tor network Shadow configurations) so that researchers and developers can benefit from our work.[4]

[2]https://github.com/privcount
[3]https://github.com/shadow
[4]https://tmodel-ccs2018.github.io

2 METHODOLOGY

We describe our measurement methodology in this section while our measurement results will be presented in Sections 3 and 4.

2.1 Measurement Goals

A primary goal of this paper is to better understand Tor traffic and its characteristics so that we can more accurately generate traffic in private Tor networks and simulators like Shadow [24]. We conduct a large Tor measurement study to help us achieve this goal. We focus our measurements on overlay network-based statistics that can also be measured in private Tor networks, including distributions of bytes per stream, streams per circuit, and circuits per client. Although some related statistics were previously reported [26], we update and expand previous measurements in order to: (i) capture the latest state of a growing Tor network; (ii) improve upon the accuracy of the previous measurement study; (iii) collect new statistics that are missing from previously reported results.

2.2 Measurement Apparatus

We measure Tor using multiple Tor relays as vantage points. Because user privacy and security of the measurement process are primary concerns in our study, we collect our measurements using a privacy-preserving distributed measurement system [26].

2.2.1 PrivCount Overview. PrivCount is an open-source privacy-preserving distributed measurement system [26] that is based on the secret-sharing variant of PrivEx [13]. PrivCount utilizes differential privacy [12] and secure aggregation in order to safely collect measurements across a set of Tor relays.

PrivCount consists of a tally server (TS), one or more share keepers (SKs), and one or more data collectors (DCs). The TS functions as a central but *untrusted* proxy that is used to facilitate communication between the SKs and DCs. The operator of a PrivCount deployment uses the TS to configure various global measurement settings, such as which statistics the DCs should collect and when they should start collecting. Each of the DCs extracts events from a Tor relay and counts their frequency in order to measure the various statistics that were configured by the TS operator. For each such statistic, each DC initializes a local counter by adding noise according to privacy parameters ϵ and δ, and also "blinds" the local counter by adding a random blinding value. The random blinding value is then secret-shared to each SK so that it can later be removed from the global count (i.e., the sum of all DC local counts). Each SK then combines the blinding values from all DCs for a given counter in order to prevent anyone from learning individual DC inputs. At the end of the measurement phase, each SK sends its summed blinding value and each DC sends its local, noisy event count to the TS; the TS sums the counts and removes the blinding values to obtain the global, noisy count.

The final global count value released by the TS is protected under (ϵ, δ)-*differential privacy* [12] because of the noise that is added by the DCs. The blinding values provide for *secure aggregation* of a measurement across all relays: as long as at least one SK is honest and secure, no one learns the individual contributions of any of the DCs to the final global count value. The blinding values also provide *forward privacy* for each DC: any compromise of a DC will

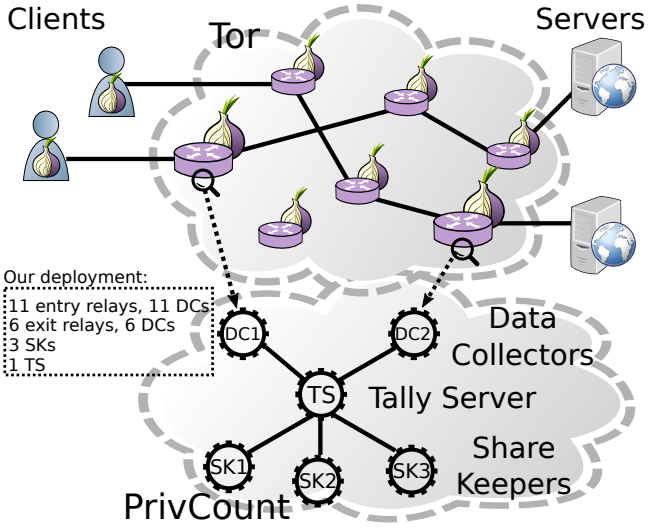


Figure 1: A simplified view of the 17 Tor relays (11 entry, 6 exit) and 21 PrivCount nodes (17 DCs, 3 SKs, and 1 TS) used in our deployment. See Table 2 for the total weights of our entry and exit relays during the periods in which they were used to collect measurements.

not leak the number of counter increments that occurred before the time of compromise. Jansen and Johnson provide further PrivCount details and proofs of security and privacy [26].

2.2.2 PrivCount Deployment. We set up a PrivCount deployment with 1 tally server, 3 share keepers, and 17 data collectors each connecting to a distinct Tor relay (we ran 6 exit and 11 non-exit relays). An overview of our deployment is shown in Figure 1. The PrivCount nodes and the Tor relays were run by 3 different operators in 3 different countries: Canada, France, and the United States. We ran the latest versions of PrivCount and PrivCount-supported Tor[5] as of the time of this writing.[6]

A primary goal of our measurement study was to improve upon the accuracy of previously reported PrivCount measurement results, which included measurement error of over 25% in many cases [26]. To help reduce error and obtain a more representative sample of Tor traffic, we increased the sampling rate of traffic that is observed by the measurement system: we ran 17 high bandwidth relays with a combined entry weight of ~1.2% and a combined exit weight of ~2.1%, compared to previous results that were collected with 7 relays providing 0.13% entry weight and 0.94% exit weight [26]. Additionally, we ensure that our exit relays will observe a representative sample of the types of traffic that Tor is used to transport by configuring a custom exit policy that is more permissive than the default exit policy. The policy we use throughout all of our measurements[7] allows traffic to ports that are commonly associated with file sharing and blocked by the default policy.

[5] A modified Tor that exports events, such as *stream ended* or *circuit ended*, to PrivCount using the Tor control protocol.
[6] Both tools are available at https://github.com/privcount.
[7] Reject *:25,*:119,*:135-139,*:445,*:563; Accept *:*

Table 1: Action Bounds in PrivCount Deployment

	Action	Bound
General	Simultaneously open entry connections	1
	Time each entry connection is open	24 Hrs.
	New circuits	144
	New streams	9,000
	File Sharing, *Other* streams	80
	Bytes transferred	10 MiB
HMM	New circuits	1
	New streams	31
	Bytes transferred	2 MiB

2.2.3 Privacy Settings. We configured our deployment to use the privacy budget allocation techniques set out by Jansen and Johnson [26], but we deviated slightly in setting the privacy parameters. We use an operator-based privacy model, wherein each operator adds enough noise across the relays they operate to fulfill the differential privacy requirements. Therefore, only one operator must remain honest (and their relays uncompromised) in order for our deployment to add the full amount of noise necessary to provide differential privacy. If there are no compromises during our measurements, then the protections on our counters will be greater since our final counters will contain $\sqrt{3}$ times as much noise as necessary. Each operator configured the privacy parameter $\delta = 10^{-3}$, which provides an upper bound on DCs choosing noise value that violates ϵ-differential privacy. For entry-based statistics, we used $\epsilon = 0.3$, and for exit-based statistics we relaxed this parameter somewhat to $\epsilon = 1.0$ in order to provide better relative accuracy for histogram-based counters while still providing adequate privacy.

Our deployment provides privacy according to the daily action bounds in Table 1, from which privacy sensitivities can be computed. For our general measurements (Section 3), we protect one simultaneously-open entry connection for the entire measurement period (the default behavior of a Tor client is to maintain a single entry connection). We protect 144 circuits of any type per day, which is enough to create one circuit every 10 minutes (the default circuit lifetime) over the length of the measurement period. According to httparchive.org, 95% of pages result in 90 or fewer connections; therefore, we protect 9,000 *web* streams (to port 80 or 443) which could be used to browse 100 pages. We protect 80 streams to ports officially associated with *file sharing* traffic[8] in order to allow for twice as many connections as peers that a new BitTorrent client typically attempts to establish [7]. Further, we observed a maximum of 74 opened connections while downloading each torrent for our BitTorrent traffic model (see Section 5.2.2). Therefore, we expect that 80 streams will protect at least one torrent download. We provide the same protections for traffic on *other* ports that are not categorized as web or file sharing, since past work has found that the vast majority of non-web traffic corresponds to file sharing traffic [8, 26, 39]. Finally, we protect 10 MiB of data transferred on streams, which covers 95% of pages according to httparchive.org. Note that histogram-based counters are sensitive to the change in

[8] 1214, [4661,4666], [6346,6429], 6699, [6881,6999], 45682, 51413

Table 2: Mean Combined Consensus Bandwidth Weights of all Relays in PrivCount Deployment during Measurements

	#	Purpose of Measurement	Weight*
Entry	1	Total clients and circuits	1.26%
	2	Circuits per client	1.13%
Exit	3	Total circuits and streams	2.13%
	4	Total bytes on streams	2.14%
	5	Streams per circuit, bytes per stream (All)	2.27%
	6	Streams per circuit, bytes per stream (Web)	2.29%
	7	Streams per circuit, bytes per stream (Other)	2.54%
	8	Hidden Markov packet model	1.49%
	9	Hidden Markov stream model	1.33%

* Weights correspond to the relay measurement position.

the number of inputs and not to the change in their values, and so we use twice the number of inputs that can change as sensitivities for histogram-based counters. (For example, we use 288 as the sensitivity for the histogram of streams per circuit, since each circuit would cause an increment to a histogram bin counter and a drop in the count of one bin would raise the count in another.)

During our hidden Markov model (HMM) measurements (Section 4), we adjust our actions bounds under the assumption that circuits in Tor are unlinkable with one another (one of Tor's design goals [11]). We protect 1 circuit during the HMM measurements, which we believe will allow for reasonable accuracy across the HMM counters. We protect 31 streams and 2 MiB which covers over 90% of circuits according to our general measurement results (see Section 3).

2.3 Measurement Process

We conducted measurements in order to update and improve the accuracy of previously-reported Tor entry and exit statistics [26] and to learn hidden Markov models of Tor traffic. Accordingly, we conducted our measurements in two main thrusts: a *general* measurement phase and a *traffic model* measurement phase. During each phase we ran several measurement periods of 24 hours each. The purpose of each measurement and the mean combined entry or exit consensus weight of our deployment during each measurement is summarized in Table 2.

During our general measurement phase, we focused most measurement periods on a collection of just a few statistics in order to avoid distributing the privacy budget too widely and to ensure we achieved the highest accuracy possible. In our earlier measurements we focused on overall totals of the number of clients, circuits, streams, and bytes and their breakdown into traffic classes. This was done to get a sense for the most important traffic classes, the distributions of which we then further investigated by collecting histogram-type counters. Histogram counters consume twice as much privacy budget as regular counters and the count is distributed among multiple bins. Therefore, we ensure that the traffic classes for which we collect histograms constitute a significant portion of Tor traffic in order to maintain reasonable accuracy.

Our general measurements were conducted between 2017-10-22 and 2017-12-01, and between 2018-01-10 and 2018-01-21. We initially used previously-published statistics as traffic estimates for the purposes of allocating our privacy budget [26], and then updated these estimates in later measurement periods using results from our earlier periods. We present in Section 3 the results from 7 general measurement periods, 2 of which were used to collect entry-based statistics (rows 1 and 2 in Table 2) and 5 of which were used to collect exit-based statistics (rows 3 through 7 in Table 2).

During our traffic model measurement phase, we focused on measuring stream and packet hidden Markov traffic models. We measure counters for each model for 14 sequential 24 hour periods. At the end of each of these periods, the counter results are fed back into the model in order to transform it to one that better fits the observed traffic. Traffic model measurements were conducted between 2018-03-19 and 2018-04-25; these measurements use only exit-based observations (rows 8 and 9 in Table 2) because we model individual TCP streams which can only be observed at Tor exit relays (due to Tor's multiplexing of streams over circuits). We present in Section 4 the results from our 28 traffic model measurement periods while also providing additional details about our model learning techniques.

2.4 Ethical Considerations and User Safety

User safety was a primary goal during the measurement process. Where applicable, we follow the techniques set out by recent privacy-preserving Tor measurement research [13, 14, 26, 38], and we utilize the state-of-the-art tool for measuring Tor in order to maintaining user privacy and security of the measurement process [26]. Additionally, we considered the safety guidelines published by the Tor research safety board[1] and discussed our plans with several board members before starting our measurements. Our tools and techniques follow many safety guidelines, including data minimization, collecting only what is safe to publish, taking reasonable security precautions (each operator had exclusive access to their machines), and limiting the granularity of data by adding noise. We believe that the benefits from our measurements outweigh the risks to user safety given our privacy and security guarantees and given that the results of this work can benefit Tor experimentation across a wide range of research areas.

3 MEASURING TRAFFIC STATISTICS

We focus our measurements on the number of clients, circuits, streams, and bytes, as well as their distributions. PrivCount counts the number of unique clients at the end of every 10 minute period in order to limit the amount of time that client IP addresses are stored in RAM. For consistency, we report all measurements as 10 minute means by dividing the daily count total by 144 (the number of 10 minute periods during each 24 hour measurement period).

3.1 Entry Statistics

Entry relays have a limited view of Tor traffic types since they do not directly observe stream information. However, they can observe clients and circuits, and hence, the distribution of circuits per client. This information is helpful in producing accurate Tor client models.

The results from our measurement of the total number of unique clients and circuits in a 10 minute interval are shown in Table 3, as well as the breakdown into *active* and *inactive* classes. From the

Table 3: The 10 minute mean entry statistics collected during period 1 (see Table 2). The error associated with the addition of privacy-preserving noise is shown with 95% confidence.

Statistic	Count ($\times 10^3$)	% of Total
Unique Clients	14.01 ± 0.513 (3.66%)	—
Active	9.978 ± 0.347 (3.48%)	71.2% ± 3.60%
Inactive	3.395 ± 0.185 (5.45%)	24.2% ± 1.59%
Circuits	679.3 ± 3.64 (0.535%)	—
Active	287.8 ± 2.94 (1.02 %)	42.4% ± 0.488%
Inactive	391.5 ± 2.15 (0.548%)	57.6% ± 0.441%

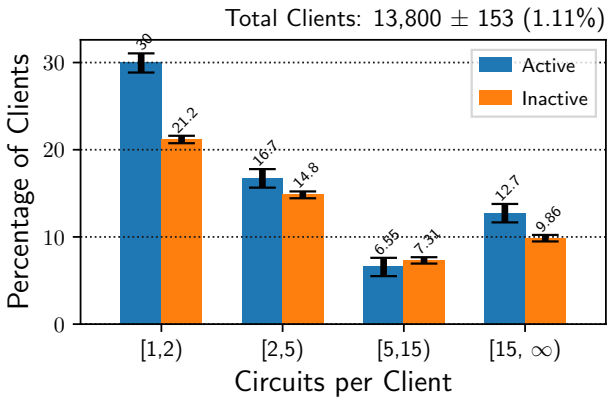


Figure 2: The 10 minute mean number of circuits per client collected during period 2 (see Table 2). The 10 minute mean number of unique clients was 13,800 ± 153 (1.11%) with 95% confidence. See Table 7 in Appendix A for a table of values.

entry perspective, PrivCount classifies a circuit as *active* if eight or more cells were sent on it (seven cells are used to construct the circuit), and otherwise classifies it as *inactive*; similarly, a client is counted as *active* if it has at least one active circuit, and *inactive* otherwise. We see from Table 3 that our entry relays observed 14,010 unique clients per 10 minute period on average, 71.2 percent of which are active while 24.2 percent are inactive. Our entry relays observed 679,300 circuits during an average 10 minutes, 42.4 percent of which are *active* and 57.6 percent of which are *inactive*. Although initially surprising, we speculate that the large number of inactive clients may be due to users who have Tor browser open but are not using it. We believe that the large number of inactive circuits are caused by Tor clients preemptively building circuits (part of Tor's design) that are never used.

Given that there is a significant amount of both active and inactive circuits, we measured the distribution of each per client using PrivCount histogram counters to better understand how clients build circuits. The results are shown in Figure 2. For both types of circuits, we observed that a single circuit per client is the most common, followed by two, three, or four circuits per client. Although we observed that 12.7 and 9.86 percent of clients build 15 or more active and inactive circuits, respectively, generally it is more likely that clients build fewer circuits. These results indicate that most

Table 4: The 10 minute mean exit circuit statistics collected during period 3 (see Table 2). The error associated with the addition of noise is shown with 95% confidence.

	Circuit Stat.	Count ($\times 10^3$)	% of Total
	Total	61.34 ± 2.20 (3.59%)	—
Total	*Active*	31.78 ± 1.06 (3.35%)	51.8% ± 2.54%
Total	*Inactive*	28.28 ± 1.14 (4.02%)	46.1% ± 2.48%
Active	*Web*	28.15 ± 0.93 (3.29%)	88.6% ± 4.16%
Active	*FileSharing*	0.850 ± 0.02 (2.66%)	2.68% ± 0.114%
Active	*Other*	4.76 ± 0.12 (2.44%)	15.0% ± 0.621%

Table 5: The 10 minute mean exit stream statistics collected during period 3 (see Table 2). The error associated with the addition of noise is shown with 95% confidence.

	Stream Stat.	Count ($\times 10^3$)	% of Total
	Total	263.5 ± 11.6 (4.39%)	—
Total	*Web*	238.4 ± 10.4 (4.38%)	90.4% ± 5.61 %
Total	*FileSharing*	1.374 ± 0.066 (4.80%)	0.52% ± 0.034%
Total	*Other*	19.91 ± 1.06 (5.34%)	7.56% ± 0.523%

Tor users only use a handful of circuits in an average 10 minutes, suggesting that many Tor Browser users are only lightly browse the web. Note that about 34 and about 47 percent of clients build zero active and inactive circuits, respectively, in an average 10 minute period, which again may be caused by idle users.

3.2 Exit Statistics

Unlike entry relays, exit relays are unable to observe clients but can observe streams. Therefore, exits have access to traffic meta-data such as the port to which a stream connects. PrivCount classifies streams to port 80 or 443 as *web*, streams to common file sharing ports as *file sharing* (see Footnote 8), and streams to unclassified ports as *other*. We initially measure traffic from all of these classes to better understand their significance. (We ignore PrivCount's *interactive* class, including SSH and IRC, as Jansen and Johnson observed an insignificant amount of interactive traffic [26].)

3.2.1 Traffic Totals. Results for our exit circuit statistics are shown in Table 4. These results show that 52.8 percent of circuits are *active* while 46.1 percent are *inactive*, which is comparable to our entry circuit statistics. We do observe about 10 times fewer circuits overall on our exits than on our entries (61,340 exit circuits compared to 679,300 entry circuits), which could be attributed to internal Tor network circuits (e.g., directory, bandwidth test, and onion service circuits) that do not utilize exit relays, circuits that fail before reaching the exit, or a bug in PrivCount. We observed that 88.6 percent of the *active* circuits carry *web* traffic, whereas only 2.68 percent carry traffic to well-known *file sharing* ports and 15 percent carry traffic on *other* ports.

Results for our exit stream statistics are shown in Table 5. Again, we observed that a large majority of streams (90.4 percent) carry

Table 6: The 10 minute mean exit byte statistics collected during period 4 (see Table 2). The error associated with the addition of noise is shown with 95% confidence.

	Byte Stat.	Count ($\times 2^{20}$)	% of Total
	Total	25,587 ± 2 (0.01%)	—
Total	*Inbound*	23,914 ± 2 (0.01%)	93.5% ± 0.009%
	Outbound	1,672 ± 0.1 (0.01%)	6.53% ± 0.001%
Total	*Web*	18,547 ± 1 (0.01%)	72.5% ± 0.007%
	FileSharing	263 ± 0.01 (0.002%)	1.03% ± 0.001%
	Other	6,744 ± 0.5 (0.01%)	26.4% ± 0.003%
Inbnd.	*Web*	17,569 ± 1 (0.01%)	73.5% ± 0.007%
	FileSharing	212 ± 0.01 (0.002%)	0.89% ± 0.001%
	Other	6,106 ± 0.4 (0.01%)	25.5% ± 0.003%
Outbnd.	*Web*	997 ± 0.03 (0.003%)	58.5% ± 0.003%
	FileSharing	51 ± 0.001 (0.003%)	3.08% ± 0.001%
	Other	637 ± 0.06 (0.01%)	38.1% ± 0.004%

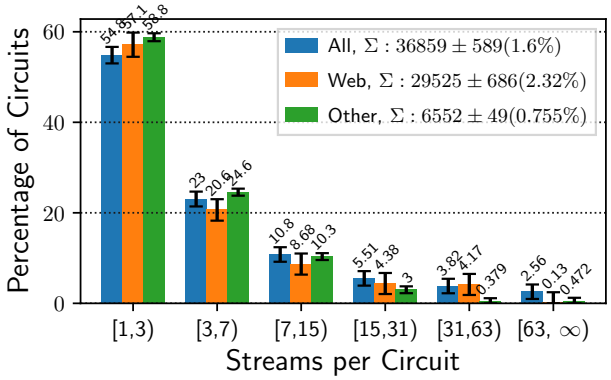


Figure 3: The distributions of streams per active circuit collected during periods 5, 6, and 7 (see Table 2). The total number of circuits of each type is shown in the legend with 95% confidence. See Table 8 in Appendix B for table of values.

traffic to *web* ports, while an insignificant number of streams (0.52 percent) carry traffic to known *file sharing* ports and a small number of streams (7.56 percent) carry *other* traffic.

Results for our exit byte statistics are shown in Table 6. Over the 24 hour measurement period, the relays in our PrivCount deployment transferred 25 GiB of exit stream data in total. Of the traffic, 93.5 percent was *inbound*, i.e., forwarded toward the circuit initiator; the remaining 6.53 percent was *outbound*, i.e., forwarded toward the Tor-external service. Of the *inbound* traffic, 73.5 percent was for *web* ports 80 and 443, while 0.89 percent was for default *file sharing* ports and 25.5 percent was for *other* ports. The share of traffic that is *web*-related fell to just 58.5 percent of *outbound* traffic, while traffic on *file sharing* ports rose to 3.08 percent and traffic on *other* ports rose to 38.1 percent.

We conclude from our exit measurements that an unsurprisingly large majority of circuits (88.6 percent), streams (90.4 percent), and bytes (72.5 percent) are associated with *web* ports 80 and 443, while most of the remaining traffic is associated with *other* ports. Past work has found that bytes on Tor corresponding to unencrypted BitTorrent were as high as 40.2 percent of observed traffic in 2008 [39] and 24.9 percent in 2010 [8]. Additionally, more recent work found some evidence that allowing more default *file sharing* ports in the relay's exit policy reduced the fraction of *web* traffic and increased the fraction of non-web traffic [26]. However, based on our observations, we conclude that traffic to default *file sharing* ports is relatively insignificant. For this reason, we focus on *web* and *other* traffic in the remainder of our exit measurements. Past work suggests that much of the traffic that we observed on *other* ports may actually be associated with BitTorrent or other file sharing, which would be consistent with the behavior of file sharing clients that choose random ports upon start-up. We don't explore this possibility further as it is out of the scope of this paper.

3.2.2 Traffic Distributions. To get a better understanding of how clients build streams and transfer bytes, we now explore distributions of streams-per-circuit and bytes-per-stream using PrivCount histogram counters. We focus on *web* and *other* traffic in this section to ensure that we capture the most significant traffic for modeling purposes, while ignoring traffic on default *file sharing* ports since it is a relatively minor contributor to Tor traffic.

The distribution of the number of streams per active circuit is shown in Figure 3. We observed that about 55 to 59 percent of active circuits carry only one or two streams and about 21 to 25 percent of circuits carry 3 to 6 streams. Another 9 to 11 percent of circuits carry 7-14 streams, and about 4-11 percent carry 15 or more streams. We were somewhat surprised that there are so few streams per circuit, given that 30 percent of popular website front pages result in 11 to 20 TCP connections and more than 45 percent of front pages result in more than 20 TCP connections according to httparchive.org. This suggests that Tor Browser users may experience the web differently than non-Tor users.

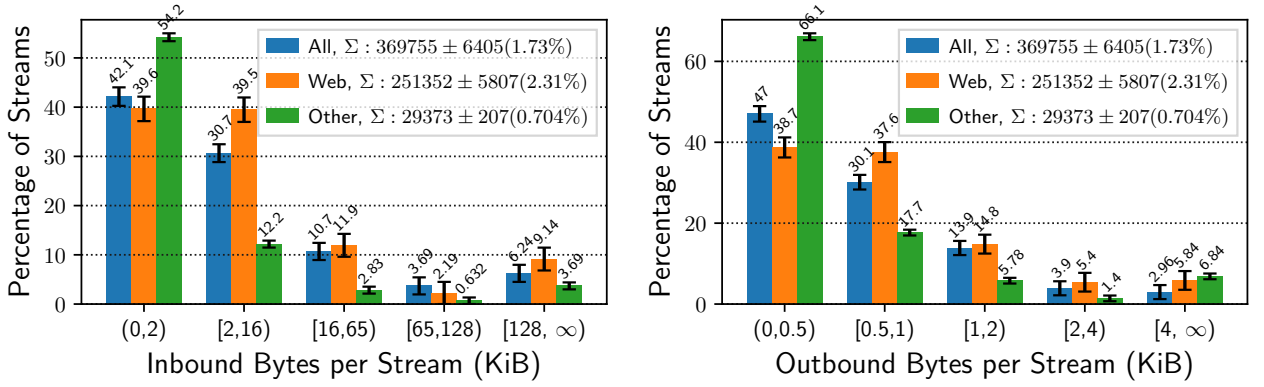
The distribution of the number of bytes per stream is shown in Figure 4, with *inbound* bytes shown in Figure 4a and *outbound* bytes shown in Figure 4b. We observed that about 70 to 80 percent of streams receive less than 16 KiB *inbound*, while about 75 to 85 percent of streams send less than 1 KiB *outbound*. We also observed a significant difference between *web* and *other* traffic: it is about 15 and 30 percent more likely that *other* streams will carry less than 2 KiB *inbound* and less than 512 bytes *outbound*, respectively, compared to *web* streams. We also observed that the highest percentage of streams sending more than 4 KiB *outbound* is 6.84 percent of *other* streams; however, the distribution of bytes per stream generally skews higher for *web* streams than for *other* streams.

4 LEARNING TRAFFIC MODELS

To model the behavior of Tor clients without explicit reference to application protocols, we model both the creation of streams and the traffic on a stream (i.e., packets) using hidden Markov models and an iterative PrivCount measurement process.

4.1 Hidden Markov Modeling

Formally speaking, a hidden Markov model (HMM) is a discrete stochastic process that maintains a state s among a state space $\mathcal{S}$. At each step i in the process, an observation o is drawn from some



(a) Inbound bytes (forwarded toward the circuit initiator) per stream. **(b) Outbound bytes (forwarded away from the circuit initiator) per stream.**

Figure 4: The distributions of bytes per stream collected during periods 5, 6, and 7 (see Table 2). The total number of streams of each type is shown in the legend and with 95% confidence. See also Tables 9 and 10 in Appendix B for full tables of values.

some observation space O according to the probability distribution $\Theta(o|s)$ conditioned (only) on the current state s_i, and a next state is drawn from $\mathcal{S}$ according to the transition distribution $P(s'|s_i)$. The first state s_0 is chosen according to the initial distribution $\pi(s)$. In our case, the states can be thought of as modeling the underlying state of the application, and the observations correspond to the arrival of streams, or packets on a stream. Since application state is opaque and traffic can be encrypted, the only properties that a relay can reliably observe are the direction (client-bound or server-bound packet) or command (open or close stream) and the time elapsed since the previous event.

Given the parameters of a stream arrival HMM, we can sample a circuit by choosing $s_0 \leftarrow \pi_{stream}(s)$, then repeatedly choosing $o_i \leftarrow \Theta_{stream}(o|s_i)$ (a delay between stream arrivals) and $s_{i+1} \leftarrow P_{stream}(s'|s_i)$ (a next application state) until a "close circuit" event occurs. Likewise, we can sample an application stream by choosing $s_0 \leftarrow \pi_{packet}(s)$, then repeatedly choosing $o_i \leftarrow \Theta_{packet}(o|s_i)$, and $s_{i+1} \leftarrow P_{packet}(s'|s_i)$ until a "close" event occurs. So producing an empirical model of Tor streams requires training a hidden Markov model to learn the parameters that best approximate real Tor streams. In general, there are several approaches to training HMMs from a set of sequences sampled from some HMM; we modified the Expectation-Maximization (EM) approach, in which an initial estimate of the parameters is used to find the sequence of states most likely to yield a sequence of observations (accomplished with the Viterbi Algorithm [15]). These states are then used to update the parameters of the HMM: each state observation distribution's parameters are chosen to maximize the probability of the observations assigned to the state, and each state's transition probabilities are chosen according to the observed frequencies. This process is then repeated, with successive parameter estimates yielding improved models of the underlying process.

To adapt the algorithm described above to the case of privately measuring Tor traffic, we slightly modified both the process of producing an initial estimate and the iteration process, as explained in the following subsections.

4.2 Bootstrapping the HMM

Training an HMM to model a process requires making some initial assumptions about the form of the model, e.g. the number of states, the connectivity between states, and the form of observation distributions. To explore reasonable choices for these assumptions, we performed several experiments using real network trace data from AS 2500, a research ISP that advertises 10 IPv4 prefixes with a total of 330,240 addresses and transfers data for real users from several academic institutions and research organizations. Because AS 2500 serves real users, the network traces that are captured are anonymized before being released as part of a network measurement project [50, 52]. The 24-hour anonymized network trace that we used included 147,962 complete TCP sessions, which we converted to "packet traces" by ignoring empty (acknowledgement-only) and duplicated packets, and treating all data-carrying packets as observations. Acknowledgements and duplicate packets were filtered because we are modeling application level behavior; the TCP stack and network emulation in Shadow will naturally (re-)produce these packets given an application stream.

4.2.1 Stream Arrivals. To obtain an initial model of stream arrivals, we hypothesized that hosts would follow a common two-state model in which one state originates streams according to an exponential distribution (representing, e.g., the sequential opening of several resources within a web page) and the other state originates streams according to a heavy-tailed distribution (encapsulating both the delay after requesting an index page and the "think time" between user page loads). Using a log-normal distribution to fit the heavy-tailed delays and adding a third state representing the closing of a circuit, we found this model a good fit for the network trace where the probability of transitioning between non-closed states was roughly 25%, the exponential process had a rate of approximately 10/second, and the log-normal process had a median delay of approximately 3 seconds.

4.2.2 Traffic. To determine the form of inter-packet delay distributions, we fit networks to a small subsample of these flows (1000)

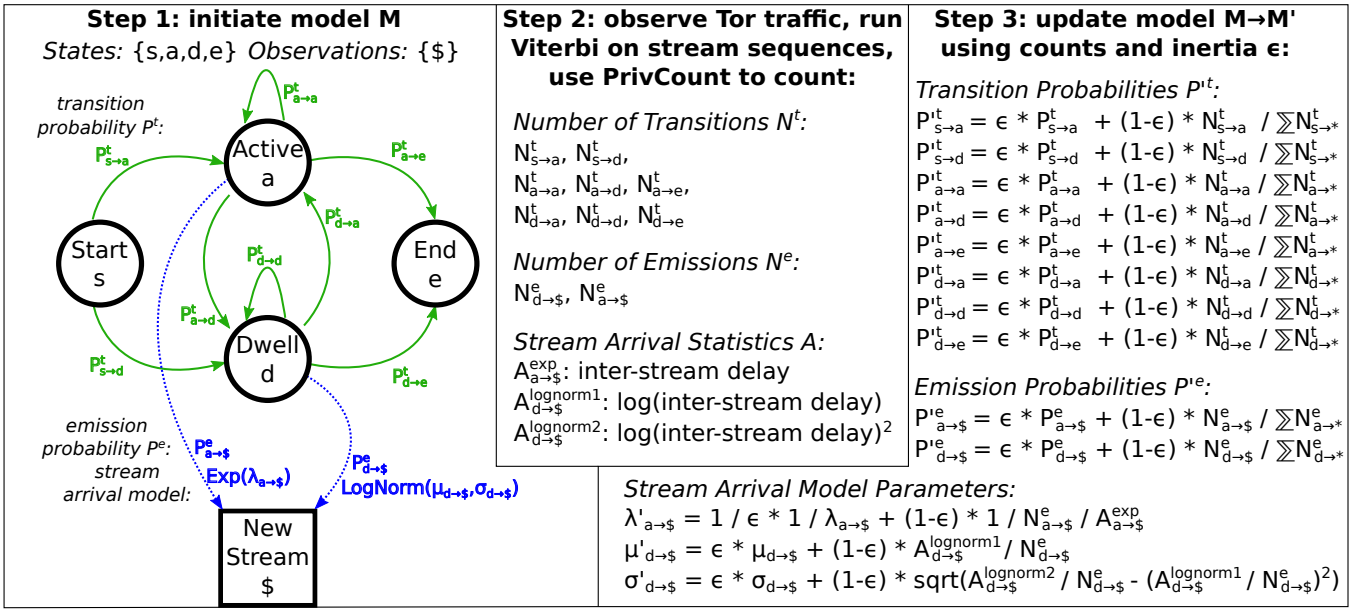


Figure 5: Overview of HMM measurement and update process using PrivCount and exemplified with our stream model.

using several different distributions. Through trial and error we found that log-normal distributions could reasonably model the delays in our trace.

To determine the architecture of the HMM (number of states and connectivity), we hypothesized that the flows could be explained by a small set of independent protocols, so we clustered the flows using an approach due to Smyth [47]. Briefly, we train an HMM to match each flow and cluster flows according to the probability their HMM assigns to other flows. We then retrain an HMM against all of the flows in the cluster. Using the "bayesian information criterion" to balance the tradeoff between the size of the resulting model and the fit to the flows, we finally produced an initial HMM with 26 states, partitioned into 3 clusters.

4.3 Tor Measurement

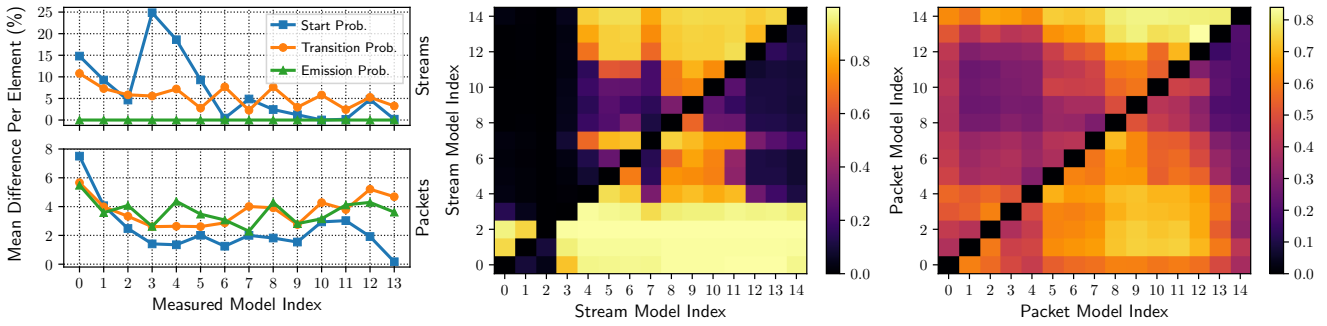
Once we have an initial model, typical EM training will repeatedly update the parameters using a single set of sequences. However in our case, we could not store the packet or stream sequences observed at our relays while preserving user privacy. Thus we modified the iterative update process to work with PrivCount as follows.

Figure 5 shows an overview of the HMM and its associated parameters, as well as the process that we used to iteratively adjust the model according to traffic observed at our Tor relays. The overview in Figure 5 uses our stream model as an example, but the process works similarly with our packet model. In each measurement period, we first initialized relays with the current HMM parameters (step 1). During the period, for each completed observation (packet or stream sequence), we used the Viterbi algorithm to compute the sequence of states that made the observation most likely. Using this state sequence, we updated sums for the number of observations of each state, the number of transitions between each pair of states, and summary statistics for the observations for each state (step 2). In the case of the stream model, this involved the sum of the delays observed in the exponential state and the sum of the log delays and squared log delays observed in the log-normal state. In the case of the packet model, we recorded sums for the number of client- and server-bound packets observed in each state, the log delay of packets observed in each state, and the squared log delay of packets observed in each state (combined with the state counts, these can be used to estimate the parameters of a log-normal distribution). At the end of a period, the aggregated counters were used to compute new parameters for the HMM (step 3): the new parameters were combined with the previous parameters using a weighted average of 0.5 to give the model "inertia". These new parameters were then used in the next measurement period.

4.4 Results

To reduce the privacy risks to users, we ran two separate measurement experiments, first performing a series of 14 iterations with the packet model, and then performing a series of 14 iterations with the stream model. Each iteration involved a 24-hour measurement period. To assess the measurement results, we were interested in two main questions: would the parameters converge to stable values over several iterations? and, would traffic patterns remain stable enough that successive iterations produced improved models?

4.4.1 Parameter Convergence. To assess the convergence of the models, we tracked the total difference in parameters between successive measurements for each iteration, that is, if the transition probability between states s and t in iteration i was $p_{st}^{(i)}$, then we computed $\sum_{s,t} |p_{st}^{(i+1)} - p_{st}^{(i)}|$ for each iteration i. Figure 6 shows the result of these differences for both the stream arrival model and the packet model. Both subplots in Figure 6a show that the change in parameters in the final model are significantly lower than



(a) Summed differences from previous model parameters per iteration.

(b) Fraction of observed stream sequences more likely under stream model x than stream model y.

(c) Fraction of observed packet sequences more likely under packet model x than packet model y.

Figure 6: Results from our Hidden Markov Model parameter convergence and model improvement analysis.

the changes in the first model, and that the differences generally decreased over measurement iterations. Note that we observed some slight oscillations between iterations, which we believe is due to PrivCount noise and possibly to some anomalous traffic that we observed during the Tor measurements.

4.4.2 Model Improvement. To determine whether successive iterations resulted in improved models, we conducted two separate experiments using all iterations of the models from our stream or packet model measurements, $M^{(0)}, M^{(1)}, \ldots, M^{(14)}$. In the stream experiment, our relay tracked stream creation events for each circuit in a 24-hour period, and for each completed circuit, the relay used the Viterbi algorithm to compute the most likely sequence of states under each model $M^{(i)}$ and the log likelihood of the observed sequence, $\ell^{(i)}$. The relay then compared each pair $\ell^{(i)}, \ell^{(j)}$ to determine which model gave the sequence a higher likelihood, and added this result to a counter. Finally, after 24 hours, these pairwise counters were taken as the output of the experiment. Intuitively, a better model for a process should assign higher likelihood to the output of the process more often; so if the models improve over iterations we should expect to see more circuits with a higher likelihood under $M^{(i)}$ than $M^{(j)}$ when $i > j$. This process was repeated with the packet model for each stream in the second experiment.

The results are shown in Figure 6. For both the stream arrival model (Figure 6b) and the packet model (Figure 6c), the later models are generally superior fits to new data than the earliest models, but we can also see that potentially anomalous measurement periods can cause some iterations to produce inferior models that then continue to improve in following iterations. Using the results of these measurements, we chose the stream and packet models that had the best performance as the basis for our Shadow experiments in the following sections (the stream and packet models at index 9).

4.4.3 Conclusions. Both measurement experiments show that a single day of Tor traffic can produce patterns that are quite anomalous compared to other 24-hour periods. An interesting subject for further research would be to expand these measurements over a longer period to reduce the effect of short-term variability in traffic. Additionally, our experiments used an "inertia" value of 0.5, meaning that our experiments gave both parameter sets equal weight in each iteration. Using a larger "inertia" value (greater than 0.5) when averaging the previous model's parameters with the latest estimates would give higher weight to the previous model's parameters and lower weight to the parameters computed from the new measurement. We expect that this may dampen the effect short-term variability in traffic and could lead the model to converge more quickly, but more work is needed to explore these issues.

5 MODELING TOR TRAFFIC

In this section, we describe modeling semantics and a tool that we designed to generate arbitrarily complex network traffic patterns. We then construct models of traffic that it can be used to generate.

5.1 Traffic Generation

We designed a set of simple traffic modeling semantics for specifying a set of actions as well as parameters for executing those actions. By using these semantics and a tool that understands them, we will be able to generate traffic patterns for most types of common applications without the need for implementing, building, installing, or configuring software components for each.

5.1.1 Modeling Semantics. We use a connected, directed *action-dependency* graph to specify an application behavior model in our framework. Each vertex in the graph corresponds to one of a small set of *actions* that are used to model behaviors, and each edge in the graph represents a *dependency* between the connected actions (i.e., the sequence in which the actions should be performed). Vertices may contain required or optional attributes that act as action input parameters and allow for customization of actions, and edges may contain attributes to adjust the meaning of the dependencies.

Valid actions in the graph include `start`, `transfer`, `model`, `pause`, and `end`. Only the `start` action is required: the graph is *walked* by starting at the `start` action and following each outgoing edge to reach the next action that should be performed. For vertices with multiple outgoing edges, the walk forks and multiple paths of execution are created and followed in parallel. This default "forking" behavior can be modified by specifying a special *weight* attribute on the edges. Only one of all outgoing edges that define a weight

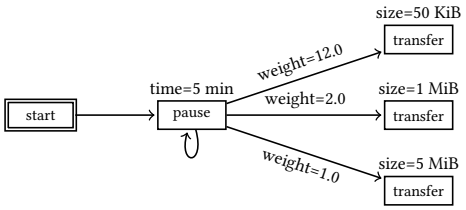


Figure 7: A simplified TGen model graph capturing the behavior of Tor's performance benchmarking process.

will be followed, where the choice is weighted according to the sum and distribution of edge weight values.

The primary means of performing a data transfer is through `transfer` and `model` actions. `Transfer` parameters include the *size* and *type* of transfer to perform. `Model` parameters include file paths to stream and packet hidden Markov models which are also represented as graphs: vertices represent states and observations and edges represent transitions and emissions (see Section 5.2.3 for more details). Pauses in the walk, between transfers or otherwise, can be specified in a `pause` action, and walk termination conditions are specified in an `end` action.

5.1.2 Traffic Generation with TGen. We developed a traffic generator application called "TGen" that uses the semantics described above to generate real network traffic. TGen parses standard graph files to extract the actions, parameters, and edge ordering. A TGen instance connects to another TGen instance and transfers data according to the parsed parameters. Various transport, transfer, and timing information is recorded during and after the transfer process so that the performance of each transfer can be later analyzed. TGen contains 6329 lines of C code and is available as open source software as part of the Shadow simulation framework [24].

TGen and our simple but powerful semantics can be used to model a wide range of behaviors. Figure 7 shows an example of a simple TGen model emulating the Tor network benchmarking process.[9] The simple model shows that a client downloads one of three differently-sized files (50 KiB, 1 MiB, and 5 MiB) every five minutes from a server, where each transfer has a weighted chance of being chosen after each five minute pause completes. Because of the modeling flexibility, The Tor Project now uses TGen to collect performance benchmarks.

5.2 Traffic Models

Tor by design fundamentally limits the ability of network participants to learn about individual users in order to protect privacy. We describe models that produce traffic either according to some generally known traffic classes and supposed application usage in order to limit privacy risks (Section 5.2.1 and Section 5.2.2) or based on our privacy-preserving PrivCount measurement results (Section 5.2.3).

5.2.1 Single File Models. Single file user models have been used almost exclusively in the Tor performance literature over the past decade. It consists of a "web" client type that downloads 320 KiB files (previously the average size of a web page) and pauses in between repeated downloads (to mimic user "think time"), and a "bulk" client type that repeatedly downloads 5 MiB files without pausing. These models are simple to understand and implement, and the number of clients of each type can be adjusted to produce different traffic distributions in a Tor simulation. However, these models may not capture Tor network conditions well. Single file models are trivial to reproduce in TGen, and we compare the efficacy of a network that uses them in Section 6.

ALGORITHM 1: Pseudocode for modeling n web sessions from over 470k pages and over 51.5m requests from HTTP Archive, simplified to show only GET request sizes (POSTs and the number of domains and TCP connections are similarly handled in our model).

Require: $D \leftarrow$ HTTP Archive Database, $n \leftarrow$ num sessions
1: $S \leftarrow$ initialize n new sessions
2: $P^1 \leftarrow$ **select** pages **from** D **sort by** page.firstRequest.responseSize
3: $B^{P^1} \leftarrow$ split(P^1, n) {split P^1 into n equally sized bins}
4: **for** i **from** 0 **to** n-1 **do**
5: $sz \leftarrow$ median(page.firstRequest.responseSize **for** page **in** $B_i^{P^1}$)
6: S_i.transfers.append(sz, first←True)
7: **end for**
8: $P^2 \leftarrow$ **select** pages **from** D **sort by** page.totalSize
9: $B^{P^2} \leftarrow$ split(P^2, n) {split P^2 into n equally sized bins}
10: **for** i **from** 0 **to** n-1 **do**
11: $m \leftarrow$ median(|page.requests| **for** page **in** $B_i^{P^2}$)
12: $R \leftarrow$ **select** requests **from** D **where** request.page **is in** $B_i^{P^2}$ **sort by** request.responseSize
13: $B^r \leftarrow$ split(R, m) {split R into m equally sized bins}
14: **for** j **from** 0 **to** m-1 **do**
15: $sz \leftarrow$ median(request.responseSize **for** request **in** B_j^r)
16: S_i.transfers.append(sz, first←False)
17: **end for**
18: **end for**

5.2.2 Protocol Models. In this section, we describe alternative "web" and "bulk" models that we designed based on traffic traces and that we believe are more realistic than the single file models.

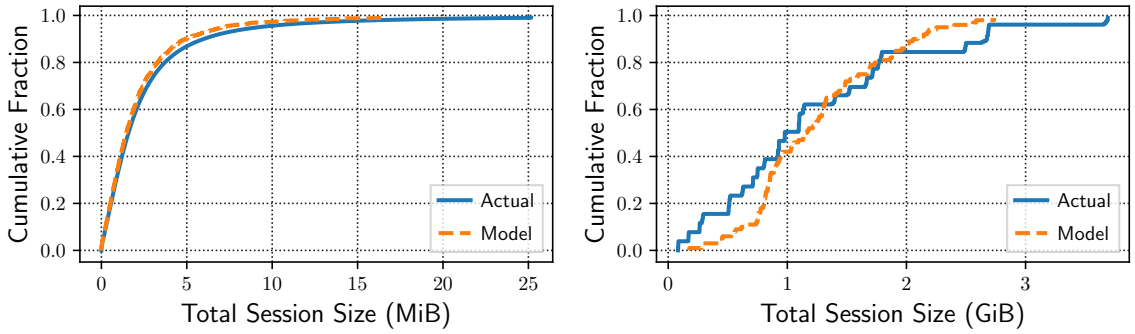
Alexa Web User Model. We design a web user model that generates traffic that is similar to real Internet website traffic. To do this we utilize data from HTTP Archive,[10] an online service that regularly fetches pages in the Alexa top 1 million sites list.[11] HTTP archive provides meta-data about each page load, which we use to build our model.

HTTP archive provides a database of *pages* and *requests*, where each page contains a single *first request* (HTML content), and many *second requests* (embedded objects). Each request is associated with a *request size*, a *response size*, and a *domain name*. We use the HTTP archive snapshot from 2017-10-16 which contains over 470 thousand pages and over 51.5 million requests in total. While we could produce a TGen graph that reproduces the transfer sizes of all of these pages and their requests, such a model would not be memory efficient (the full HTTP archive database consumes 52 GiB of persistent storage space). Instead, we model the page loads as a

[9] https://metrics.torproject.org/torperf.html

[10] http://httparchive.org

[11] http://www.alexa.com/topsites



(a) HTTP Archive Model **(b) BitTorrent Model**

Figure 8: Total session size distribution in our general user models compared to actual session sizes in the full datasets.

smaller number multiple-transfer *sessions* that maintain the overall distributions of page, request, and response sizes.

Our modeling approach is detailed in Algorithm 1. To create n sessions, we first get a list of all of the first requests on pages, sorted by their response sizes. We then split this list into n equally sized bins, and take the median of the response sizes in each bin, and use them as the first transfer sizes in our sessions (see lines 2-7). This *median-binning* strategy ensures that the distribution of our first request sizes *best* fits the distribution of request sizes in the original dataset. We use a similar median-binning approach to compute the number of requests for each session in lines 8-11, and their sizes in lines 12-15. We show the distribution of the modeled total session sizes compared to actual total page sizes in Figure 8a; a *Kolmogorov–Smirnov* test of the distributions yields a result of 0.045 ($p < 0.05$) which indicates that our model approximates the dataset reasonably well.

BitTorrent Bulk User Model. We also design a BitTorrent user model that generates traffic similar to real-world BitTorrent traffic. We use the same general strategy as for the web user model. We were unable to find a dataset containing BitTorrent transfer information, so we generated our own.

We sourced 30 torrent files from a wide variety of open source Linux distributions and free software websites. Each torrent was loaded one at a time into a version of the popular BitTorrent client rTorrent[12] that had been modified to export useful events regarding connections and incoming/outgoing pieces. Each torrent was allowed to completely finish downloading before the next started. The torrents were downloaded while throttled to 25, 50, 75, and 100 Mbit/s for a total of 4 samples per torrent. We performed this collection process on three different machines located in the United Kingdom, California, and Washington DC. The machines had symmetric Internet connections limited to 10 Gbit/s, 1 Gbit/s, and 50 Mbit/s respectively. We successfully collected 309 samples of BitTorrent traffic in total during 2017-12 and 2018-01 with a wide range of total torrent sizes, BitTorrent swarm sizes, and effective maximum bandwidth rates.

To extract n representative sessions from this data, we used a process similar to Algorithm 1. We start by sorting the sessions by the number of connections that they made or received (like line 2). The sessions are then binned (like line 3) and the median number of connections in each bin becomes the number used for one of the n representative sessions (like lines 4-7).

From here, each of the n sessions has been assigned a number of connections, but not an amount to to send, an amount to receive, or a connection start time. To determine how much each of session n_i's connections should send, we list all connections made in the sessions in the ith bin, sort them by amount sent, and perform median-binning. The analogous is done to determine an amount to receive. We use median-binning to calculate a start time, but we shuffle the resulting times such that the connection that sends and receives the least does not also start the soonest.

The resulting distribution of modeled session sizes, where session size is the sum of the bytes received across all connections, is plotted against actual session sizes in Figure 8b; a *Kolmogorov–Smirnov* test of the distributions yields a result of 0.22 ($p < 0.005$) which indicates that our model approximates the dataset reasonably well.

5.2.3 PrivCount Model. Our PrivCount model is constructed using a combination of the measurement results that we presented in Section 3 and Section 4. Most importantly, each client generates streams and packets in the PrivCount model according to the HMMs from Section 4. We specify the HMM graph files as parameters for a `model` action type in the TGen action graph (see Section 5.1.1). For each such `model` action, we traverse the states in the stream HMM graph to generate stream observations and we generate packet schedules for each stream using the packet HMM graph. Delays between streams and packets are sampled from the probability distributions associated with the emission of each observation (parameters for which are encoded in the respective HMM graphs). The traffic generation process continues until we reach the close state in the stream HMM; once all of the generated stream transfers finish, the `model` action is complete and TGen moves on to the next action in its action graph.

We use our measurements from Section 3 to understand how many clients and circuits to create. In particular, we create a number of active clients according to our entry measurements from Table 3 while ignoring inactive clients. We scale our measurements

[12] https://rakshasa.github.io/rtorrent

according to the combined bandwidth fraction of the relays in our PrivCount deployment during the measurement (see Table 2) and the fraction of the Tor network that we are attempting to simulate. For each active client, we sample a number of active circuits c from the circuits-per-client distribution in Figure 2. We create c `model` actions in each client TGen graph, and use Tor's `IsolateSOCKSAuth` configuration option and a unique SOCKS username and password combination for each `model` action when creating the connections to Tor SOCKS proxy to ensure that a unique circuit is used for streams generated by each action (this is same technique that is used by Tor Browser Bundle to force streams onto different circuits). Each client starts each of its c `model` actions at $t \leftarrow \frac{600}{c}$ seconds from the previous, such that the traffic generation process for each `model` action starts at a time uniformly distributed throughout each 10 minute interval.

6 EVALUATION

In the previous sections, we measured Tor and described traffic models that can be used to generate traffic in private Tor networks. In this section, we evaluate the fidelity of private Tor networks that generate traffic according to our models.

6.1 Experimental Setup

We use Shadow [24] to evaluate and compare our traffic models. Shadow is a network simulator that runs real applications, including Tor and TGen (described in Section 5.1.2). Shadow uses a network graph to model Internet paths, including latency and bandwidth rates, and runs a private Tor network over this Internet model. We created an updated Shadow network model since the most recent Shadow network graph available was created in 2013 [23].

6.1.1 Network Model. We are interested in modeling network latency between all pairs of nodes that run in Shadow. To ensure that our model is general enough to be useful for a variety of experiments and for future work, we create a city-based latency map using Internet `ping` measurements.

We use RIPE Atlas[13] to perform and collect `ping` measurements. Atlas provides vantage points called probes from which measurements can be scheduled and collected. We first measured all 8,428 available Atlas probes and found that 2,993 of them responded to `ping`. To reduce the overall number of required measurements, we use only one representative probe per city (as assigned by Maxmind geoip[14]) which had the highest uptime of those probes that are assigned to the same city. This paring process left us with 1,813 probes, among all pairs of which we scheduled 3 ping measurements in Atlas. Of the $\frac{n \cdot (n-1)}{2} = 1,642,578$ total pairs of probes, we obtained at least one successful latency estimate for 1,245,948 (76%) of the pairs and no successful estimate for 396,630 (24%) of the pairs.

Every vertex in our network graph represents one of the cities for which we obtained a successful `ping` measurement; each such vertex is assigned an IP address, city code, and country code, as well as city upstream and downstream bandwidth rates that we parsed from speedtest.[15] For every pair of vertices (i.e., cities), we created an edge in our graph and assigned it a latency according to the mean of the successful pings between the pair; if no successful measurement was available, then we take the mean of all pings between the two countries in which the cities were located. Similarly, we used the mean bandwidth rates for the country in which the city was located for any city for which speedtest did not provide city bandwidth rates. Using this approach, we always favor the most specific data that is available.

We were unable to find a city-based packet loss data set. Instead, we assign each edge e a fractional packet loss rate p_e that increases linearly with the latency l_e assigned to the edge, such that $p_e = 0.015 \cdot l_e/300$. This loss model has been shown to yield reasonable results when running Tor in Shadow [30].

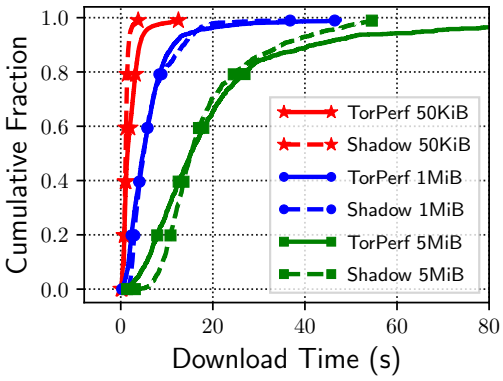


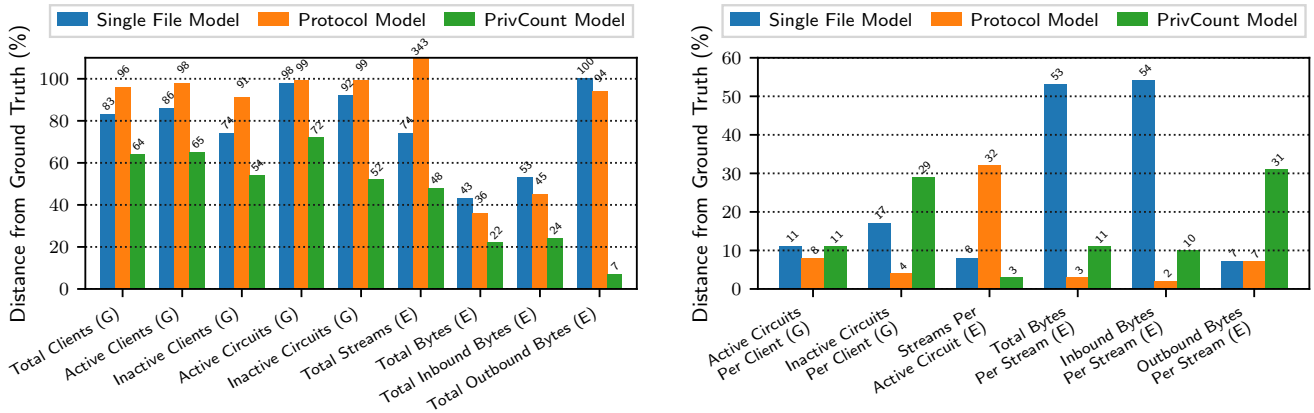
Figure 9: Shadow and TorPerf performance benchmarks.

6.1.2 Tor Host Model. We created three Tor host models for Shadow, one for each of the traffic models from Section 5.2. We used the standard network generation tools provided with Shadow in order to generate the *single file* model. This network was configured with 2000 Tor relays, 5000 TGen servers, and 60,000 TGen clients. Of the TGen clients, 57,327 were configured to run the single file "web" model, 1773 were configured to run the single file "bulk" model, and 900 were configured to emulate the TorPerf process (see Figure 7). The resulting network configuration yields similar performance as TorPerf benchmarks from the public network as shown in Figure 9.

In our *protocol* model we replace the "web" and "bulk" single file TGen models from above with the more complex HTTP and BitTorrent models described in Section 5.2.2. We also reduce the number of clients: we create 4.8 times fewer "web" clients (i.e., 11,943) since the median session size in our new model is 4.8 times greater than the single file model (1.5 MiB vs. 320 KiB), and we create half as many "bulk" clients since we expect the new multi-stream BitTorrent model to achieve twice the throughput of the single file model (due to Tor's flow control, i.e., circuit and stream windows [2]).

In our *PrivCount* model, we replace the "web" and "bulk" single file TGen models with HMM clients that follow the traffic generation process as described in Section 5.2.3. We run half as many active HMM clients as scaling our measurements in Table 3 dictates due to resource constraints (each Tor client consumes ~10-20 MiB of RAM): we configure 128,519 TGen clients that each generate

[13] https://atlas.ripe.net
[14] https://www.maxmind.com
[15] http://www.speedtest.net



(a) Single Counter Type

(b) Histogram Counter Type

Figure 10: Wasserstein distance (a.k.a., earth mover's distance) between PrivCount measurements as a percentage of ground truth, where a (G) indicates a measurement taken from an entry relay position and an (E) indicates a measurement taken from an exit relay position. (10a) The cumulative percentage distance across all nine single counters was 703% for the single file model, 1001% for the protocol model, and 408% for the PrivCount model. (10b) The cumulative percentage distance across all six histogram counters was 150% for the single file model, 56% for the protocol model, and 95% for the PrivCount model.

twice as much traffic (and twice as many circuits) as explained in in Section 5.2.3 using our HMM traffic generation process.

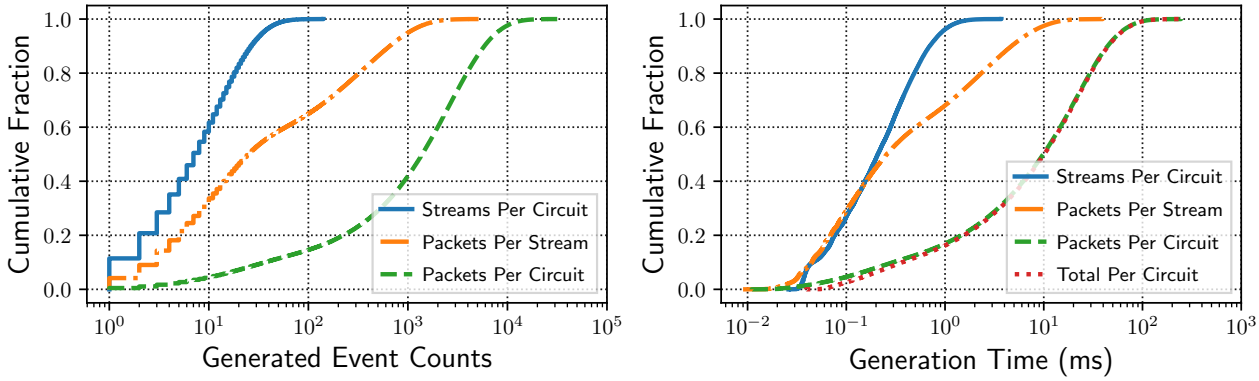
6.2 Results

We run our Tor networks in Shadow using each of the three client models outlined above, and using a version of Tor that exports PrivCount events that each relay logs to a file throughout the simulation. Following each experiment, we configure a local PrivCount deployment that consumes the logged events from our files rather than from relays' control ports as is PrivCount's standard mode of operation. This process allows us to directly compare our measurement results from the public Tor network (i.e., ground truth) with those from our Shadow simulations across a range of statistics.

6.2.1 Distance. We compare the measurement results from each Shadow experiment to the ground truth Tor measurements using Wasserstein distance (a.k.a., earth mover's distance) as a metric. Under this metric, the computed distance represents the minimum amount of "dirt" that needs to be moved for each measurement in order to yield an identical result to the ground truth measurement. Each unit has equal weight for single counters, while for histograms the distance for each bin i with a corresponding bin range $[L_i, R_i)$ is weighted as $(L_i + R_i)/2$.

We present the Wasserstein distances for each Shadow experiment as percentages of the total ground truth values in Figure 10. Figure 10a shows that our PrivCount model is closer to Tor for all single-type counters that we measured. The cumulative percentage distance across all nine single counters was 703% for the single file model, 1001% for the protocol model, and 408% for the PrivCount model. The highest distance values for the single file and protocol models is attributed to the network producing significantly fewer inactive clients, circuits of all types, and outbound bytes, and the protocol model produced 343% more streams than indicated by our ground truth measurements. The PrivCount model improves upon these by incorporating our measurement results into the model. Figure 10b shows that the protocol and PrivCount models are similarly close to Tor for histogram-type counters: the cumulative percentage distance across all six histogram counters was 150% for the single file model, 56% for the protocol model, and 95% for the PrivCount model. We find that the single file model produces streams with significantly more inbound bytes than we measured in Tor (and the constant file sizes skew the distribution), while the protocol model creates more streams per circuit than we observed in Tor. We find that the largest distance in the PrivCount model was due to the creation of too many inactive circuits and too many outbound bytes per stream. We believe that the larger number of inactive circuits was a result of more clients building more preemptive circuits than in the other models; the PrivCount model used about 10 times the number of clients as the protocol model, and about twice as many clients as the single file model, and the distance associated with inactive clients scales similarly. The larger number of outbound bytes per stream is an artifact of our TGen HMM implementation: the server-to-client packet delay sequence is generated on the client side and sent to the server upon stream creation, yielding more client-to-server bytes than usual. Future work should consider an implementation that removes this artifact.

6.2.2 Performance. Although the PrivCount model produces a network that is "closer" to Tor, we measured the performance of our HMM generator process to better understand its overhead. We used our stream model to generate 439,344 "circuits" (here, each "circuit" corresponds to a sequence of inter-stream delays in our experiments), and we used our packet model to generate 4,980,038 packet sequences (i.e., one inter-packet delay sequence for each stream in each circuit). We measured the time to generate each sequence, the total time to generate all packets and streams for each circuit, and the generate event counts. Note that we measure the time to generate the inter-packet and inter-stream delay sequences,



(a) Number of Generated HMM Events **(b) Time to Generate HMM Events**

Figure 11: Performance overhead of our HMM generator processes.

but not the time to create packets and streams or transfer traffic. The performance results are shown in Figure 11.

Our generator continues to generate streams for a circuit until the stream model emits an end event, and it generates packets for a stream until the packet model emits an end event. The number and distribution of such events that were generated in our performance experiment are shown in Figure 11a. Our generator generated fewer than 165 streams per circuit across all generated circuits. We found that our generator did not generate more than 7,211 packets (~10.5 MiB) for a single stream, and did not generate more than 35,568 packets (51.9 MiB) for all streams in a circuit.

Figure 11b shows the time to generate HMM events. We found that the maximum time to generate all of the streams in a circuit, for all circuits, was 4 milliseconds. We also found that the maximum time to generate a packet sequence for each stream was 52 milliseconds, and the maximum time to generate all packet sequences for all streams in the same circuit was 274 milliseconds. Similarly, the maximum total generator time for all streams and packets in a circuit was 278 milliseconds. We believe that the generator times are negligible when compared to the time to send and receive network traffic (which happens as a result of the generated events).

Finally, we found that the PrivCount model requires more computational and memory resources to run in Shadow compared to the single file and protocol models. We believe this is because the PrivCount model contains more clients than the other models. Given our results, we suspect that inactive clients can be excluded to save resources without significant detrimental effect. An area of further research would be to explore how running fewer but busier clients affects experimental accuracy, and how changes in experimental accuracy affects research conclusions.

7 RELATED WORK

This work focuses on the measuring and modeling of Tor traffic.

7.1 Measuring Tor

A measurement process is necessary for distributed system operators that wish to understand the usage and performance of their system. A simple approach taken in early Tor measurement studies from McCoy *et al.* [39] and Chaabane *et al.* [8] was to directly log traffic and later analyze it to extract interesting statistics. Because these studies recorded and manually analyzed sensitive data (including packet headers and some fraction of packet payloads), they were met with strong criticism from the privacy community. Following these studies, Soghoian raised ethical and legal questions and called for better standards for safer Tor research [49].

Loesing provided guidelines for safely measuring anonymity networks and conducted a privacy-preserving Tor measurement study of the number of connecting clients and amount of exit traffic by port [37]. The privacy techniques include simple rounding and per-relay aggregation over time; secure aggregation *across* relays is not considered and there is no rigorous analysis of privacy guarantees, both of which our measurement study benefits from due to our use of PrivCount [26].

Despite Soghoian's and Loesing's work, a traffic logging approach was still later used by Ling *et al.* [35] to measure and classify malicious traffic on Tor using the snort intrusion detection system. More recently, Owen and Savage collected a list of unique hidden service addresses by running a large number of hidden service directories and recording hidden service lookups [42]. These direct logging studies are illuminating but are ethically questionable since they provide no privacy protections.

There has been considerable advances in recent years in privacy-preserving Tor measurement techniques and tools. Elahi *et al.* first designed secret-sharing and distributed decryption variants of PrivEx, a traffic statistics measurement system based on distributed differential privacy and secure multiparty computation [13]. Jansen and Johnson extended the secret-sharing variant of PrivEx to design and develop PrivCount, which they used to measure a wide range of Tor traffic statistics [26]. Jansen *et al.* extended PrivCount to support the classification of circuits and webpages, and used it to measure the popularity of hidden services [28]. Our work also extends PrivCount to support the measurement and processing of hidden markov traffic models. Finally, Mani and Sherr developed

Historϵ, a distributed measurement system that is robust to manipulation by malicious data collectors [38], while Fenske *et al.* designed Private Set-union Cardinality, a cryptographic protocol for aggregating the count of *unique* items across data collectors [14].

7.2 Modeling Tor

Early Tor client models were created for use in single-purpose discrete-event Tor simulators that explore the performance benefits of Tor incentive schemes [25, 41]. Variants of these client models that utilized some of the Tor-specific statistics measured by McCoy *et al.* [39] were later adopted into higher-fidelity Tor experimentation tools [6, 22, 24]. These early models, which correspond to our single file models from Section 5.2.1, became somewhat of an unofficial standard in Tor performance research because: (i) they were based on actual Tor measurements; and (ii) it was difficult to safely and ethically collect new Tor measurements that could be used in new models. We seek to better understand the accuracy and fidelity of these previous models while also using recent privacy-preserving measurement tools to produce newer, data-driven models that are more representative of the current Tor network.

Models of Tor network structure, including Internet paths, latency, packet loss rates, and relay and client types and distribution, were first rigorously set out by Jansen *et al.* [22]. The network topologies of these initial models were independently improved by Wacek *et al.* [51], and later integrated and validated for Shadow [24] by Jansen *et al.* [23]. We produce the most recent model of network topology using real Internet measurements from RIPE Atlas and the most recent bandwidth data available from speedtest.net. We believe that our topology yields the best trade-off between fidelity, measurement overhead, and processing delays during simulation of any previous model.

8 CONCLUSION

In this paper, we conducted a significant general measurement study of Tor, measuring clients, circuits, streams, bytes, and their distributions for modeling purposes. Additionally, we collected measurements that we used to dynamically learn hidden Markov stream arrival and packet models. Using our measurements, we designed a traffic generation model for private Tor networks and demonstrated that our model yields a more realistic network than previous and alternative models. We identified interesting areas of future work, including the measurement of HMM stream and packet models over time periods longer than 24 hours and exploring how network accuracy affects research conclusions across a range of network scales.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their feedback and suggestions to improve the paper. We thank Tim Wilson-Brown for running some of the Tor relays and PrivCount nodes that we used to measure Tor, Ryan Wails for the discussion about modeling Internet latency, and Ryan Wails and Phillip Winter for donating RIPE Atlas credits that we used to run Atlas measurements. This work has been partially supported by the Office of Naval Research, the National Science Foundation under grant numbers CNS-1527401 and CNS-1314637, and the Department of Homeland Security Science and Technology Directorate, Homeland Security Advanced Research Projects Agency, Cyber Security Division under agreement number FTCY1500057. The views expressed in this work are strictly those of the authors and do not necessarily reflect the official policy or position of any employer or funding agency.

REFERENCES

[1] Mashael AlSabah, Kevin Bauer, Tariq Elahi, and Ian Goldberg. The path less travelled: Overcoming Tor's bottlenecks with traffic splitting. In *Privacy Enhancing Technologies Symposium (PETS)*, pages 143–163. Springer, 2013.

[2] Mashael AlSabah, Kevin Bauer, Ian Goldberg, Dirk Grunwald, Damon McCoy, Stefan Savage, and Geoffrey M Voelker. Defenestrator: Throwing out windows in Tor. In *Privacy Enhancing Technologies Symposium (PETS)*, pages 134–154. Springer, 2011.

[3] Mashael AlSabah and Ian Goldberg. PCTCP: Per-circuit tcp-over-ipsec transport for anonymous communication overlay networks. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.

[4] Mashael AlSabah and Ian Goldberg. Performance and security improvements for Tor: A survey. *ACM Computing Surveys (CSUR)*, 49(2):32, 2016.

[5] Armon Barton and Matthew Wright. DeNASA: Destination-naive AS-awareness in anonymous communications. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2016(4):356–372, 2016.

[6] Kevin S Bauer, Micah Sherr, and Dirk Grunwald. ExperimenTor: A testbed for safe and realistic Tor experimentation. In *USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, 2011.

[7] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a bittorrent networks performance mechanisms. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2006.

[8] A. Chaabane, P. Manils, and M.A. Kaafar. Digging into anonymous traffic: A deep analysis of the Tor anonymizing network. In *IEEE Network and System Security (NSS)*, 2010.

[9] Bernd Conrad and Fatemeh Shirazi. Analyzing the effectiveness of dos attacks on tor. In *Proceedings of the 7th International Conference on Security of Information and Networks*, page 355. ACM, 2014.

[10] S. Dahal, Junghee Lee, Jungmin Kang, and Seokjoo Shin. Analysis on end-to-end node selection probability in tor network. In *2015 International Conference on Information Networking (ICOIN)*, pages 46–50, Jan 2015.

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

[12] Cynthia Dwork. Differential privacy. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2006.

[13] Tariq Elahi, George Danezis, and Ian Goldberg. PrivEx: Private collection of traffic statistics for anonymous communication networks. In *ACM Conference on Computer and Communications Security (CCS)*, 2014. See also git://git-crysp.uwaterloo.ca/privex.

[14] Ellis Fenske, Akshaya Mani, Aaron Johnson, and Micah Sherr. Distributed measurement with private set-union cardinality. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[15] G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, March 1973.

[16] John Geddes, Rob Jansen, and Nicholas Hopper. How low can you go: Balancing performance with anonymity in tor. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 164–184. Springer, 2013.

[17] John Geddes, Rob Jansen, and Nicholas Hopper. IMUX: Managing Tor connections from two to infinity, and beyond. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 181–190. ACM, 2014.

[18] John Geddes, Mike Schliep, and Nicholas Hopper. Abra cadabra: Magically increasing network utilization in tor by avoiding bottlenecks. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, pages 165–176. ACM, 2016.

[19] Deepika Gopal and Nadia Heninger. Torchestra: Reducing interactive traffic delays over Tor. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2012.

[20] Nicholas Hopper. Challenges in protecting tor hidden services from botnet abuse. In *Financial Cryptography and Data Security (FC)*, pages 316–325. Springer, 2014.

[21] Mohsen Imani, Armon Barton, and Matthew Wright. Guard sets in tor using as relationships. *Proceedings on Privacy Enhancing Technologies*, 2018(1):145–165, 2018.

[22] Rob Jansen, Kevin Bauer, Nicholas Hopper, and Roger Dingledine. Methodically modeling the Tor network. In *USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, 2012.

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

[24] Rob Jansen and Nicholas Hopper. Shadow: Running Tor in a box for accurate and efficient experimentation. In *Network and Distributed System Security Symposium (NDSS)*, 2012. See also https://shadow.github.io.

[25] Rob Jansen, Nicholas Hopper, and Yongdae Kim. Recruiting new Tor relays with BRAIDS. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.

[26] Rob Jansen and Aaron Johnson. Safely measuring Tor. In *ACM Conference on Computer and Communications Security (CCS)*, 2016. See also https://github.com/privcount.

[27] Rob Jansen, Aaron Johnson, and Paul Syverson. LIRA: Lightweight incentivized routing for anonymity. In *Network and Distributed System Security Symposium (NDSS)*, 2013.

[28] Rob Jansen, Marc Juarez, Rafael Galvez, Tariq Elahi, and Claudia Diaz. Inside Job: Applying traffic analysis to measure Tor from within. In *Network and Distributed System Security Symposium (NDSS)*, 2018.

[29] Rob Jansen, Paul F Syverson, and Nicholas Hopper. Throttling Tor bandwidth parasites. In *USENIX Security Symposium (USENIX)*, 2012.

[30] Rob Jansen and Matthew Traudt. Tor's been KIST: A case study of transitioning Tor research to practice. *CoRR (arXiv)*, abs/1709.01044, 2017.

[31] Rob Jansen, Florian Tschorsch, Aaron Johnson, and Björn Scheuermann. The sniper attack: Anonymously deanonymizing and disabling the Tor network. In *Network and Distributed System Security Symposium (NDSS)*, 2014.

[32] Aaron Johnson, Rob Jansen, Nicholas Hopper, Aaron Segal, and Paul Syverson. PeerFlow: Secure load balancing in Tor. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2017(2):74–94, 2017.

[33] Aaron Johnson, Rob Jansen, Aaron D Jaggard, Joan Feigenbaum, and Paul Syverson. Avoiding the man on the wire: Improving Tor's security with trust-aware path selection. In *Network and Distributed System Security Symposium (NDSS)*, 2017.

[34] Dong Lin, Micah Sherr, and Boon Thau Loo. Scalable and anonymous group communication with MTor. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2016(2):22–39, 2016.

[35] Zhen Ling, Junzhou Luo, Kui Wu, Wei Yu, and Xinwen Fu. TorWard: Discovery of malicious traffic over Tor. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2014.

[36] Zhuotao Liu, Yushan Liu, Philipp Winter, Prateek Mittal, and Yih-Chun Hu. TorPolice: Towards enforcing service-defined access policies for anonymous communication in the Tor network. In *International Conference on Network Protocols*. IEEE, 2017.

[37] Karsten Loesing, Steven J. Murdoch, and Roger Dingledine. A case study on measuring statistical data in the Tor anonymity network. In *Financial Cryptography and Data Security (FC)*, 2010. See also https://metrics.torproject.org.

[38] Akshaya Mani and Micah Sherr. HisTorε: Differentially private and robust statistics collection for Tor. In *Network and Distributed System Security Symposium (NDSS)*, 2017.

[39] Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Shining light in dark places: Understanding the Tor network. In *Privacy Enhancing Technologies Symposium (PETS)*, 2008.

[40] W. Brad Moore, Chris Wacek, and Micah Sherr. Exploring the potential benefits of expanded rate limiting in Tor: Slow and steady wins the race with tortoise. In *Annual Computer Security Applications Conference (ACSAC)*, ACSAC '11, 2011.

[41] Tsuen-Wan "Johnny" Ngan, Roger Dingledine, and Dan S. Wallach. Building incentives into Tor. In *Financial Cryptography and Data Security (FC)*, 2010.

[42] Gareth Owen and Nick Savage. Empirical analysis of Tor hidden services. *IET Information Security*, 10(3):113–118, 2016.

[43] Florentin Rochet and Olivier Pereira. Waterfilling: Balancing the tor network with maximum diversity. *Proceedings on Privacy Enhancing Technologies*, 2017(2):4–22, 2017.

[44] Florentin Rochet and Olivier Pereira. Dropping on the edge: Flexibility and traffic confirmation in onion routing protocols. *Proceedings on Privacy Enhancing Technologies*, 2018(2):27–46, 2018.

[45] Fatemeh Shirazi, Claudia Diaz, and Joss Wright. Towards measuring resilience in anonymous communication networks. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*, pages 95–99. ACM, 2015.

[46] Fatemeh Shirazi, Matthias Goehring, and Claudia Diaz. Tor experimentation tools. In *IEEE International Workshop on Privacy Engineering (IWPE)*, IWPE '15, pages 206–213, 2015.

[47] Padhraic Smyth. Clustering sequences with hidden markov models. In *Advances in neural information processing systems*, pages 648–654, 1997.

[48] Robin Snader and Nikita Borisov. Eigenspeed: Secure peer-to-peer bandwidth evaluation. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2009.

[49] Christopher Soghoian. Enforced community standards for research on users of the Tor anonymity network. In *Workshop on Ethics in Computer Security Research (WECSR)*, 2011.

[50] CSL Sony and Kenjiro Cho. Traffic data repository at the wide project. In *Proceedings of USENIX 2000 Annual Technical Conference: FREENIX Track*, pages 263–270, 2000.

[51] Chris Wacek, Henry Tan, Kevin Bauer, and Micah Sherr. An Empirical Evaluation of Relay Selection in Tor. In *Network and Distributed System Security Symposium (NDSS)*, 2013.

[52] The WIDE Project. Traffic trace info, 2016-07-08. http://mawi.wide.ad.jp/mawi/samplepoint-F/2016/201607081400.html, July 2016.

[53] Lei Yang and Fengjun Li. Enhancing traffic analysis resistance for tor hidden services with multipath routing. In *International Conference on Security and Privacy in Communication Systems*, pages 367–384. Springer, 2015.

[54] Lei Yang and Fengjun Li. mtor: A multipath tor routing beyond bandwidth throttling. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 479–487, Sept 2015.

APPENDIX

A ENTRY STATISTICS

Here we provide tables of entry statistics for graphical figures that were included in the main body of the paper. Table 7 shows values of entry statistics that were shown in the plots in Figure 2.

Table 7: The 10 minute mean number of circuits per client collected during period 2 (see Table 2). The 10 minute mean number of unique clients was 13,800 ± 153 (1.11%) with 95% confidence. See also Figure 2.

Hist. Bin		Count ($\times 10^3$)	% of Total
[1,2)	*Active*	4.13 ± 0.144 (3.50%)	30.0% ± 1.10%
	Inactive	2.92 ± 0.049 (1.68%)	21.2% ± 0.426%
[2,5)	*Active*	2.31 ± 0.144 (6.26%)	16.7% ± 1.06%
	Inactive	2.05 ± 0.049 (2.40%)	14.8% ± 0.392%
[5,15)	*Active*	0.904 ± 0.144 (16.0%)	6.55% ± 1.05%
	Inactive	1.01 ± 0.049 (4.87%)	7.31% ± 0.365%
[15,∞)	*Active*	1.76 ± 0.144 (8.23%)	12.7% ± 1.06%
	Inactive	1.37 ± 0.049 (3.62%)	9.86% ± 0.373%

B EXIT STATISTICS

Here we provide tables of exit statistics for graphical figures that were included in the main body of the paper, as well as a graphical figure that was not included in the main body of the paper for space reasons. Table 8 shows values of exit statistics that were shown in the plots in Figure 3, Table 9 provides values of exit statistics that were shown in the plots in Figure 4a, and Table 10 provides values of exit statistics that were shown in the plots in Figure 4b. The distribution of total bytes per stream is shown in Figure 12 and the full values are shown in Table 11.

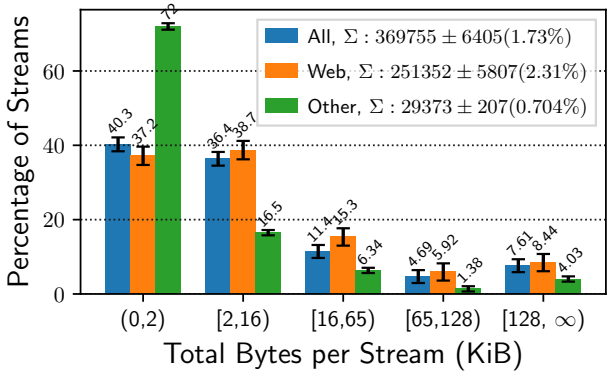


Figure 12: The distributions of total bytes per stream (inbound + outbound) collected during periods 5, 6, and 7 (see Table 2). See also Table 11 in Appendix B for full tables of values.

Table 8: The 10 minute mean number of streams per active circuit collected during periods 5, 6, and 7 (see Table 2). See also Figure 3. (See the legend in Figure 3 for the total number of *all*, *web*, and *other* circuits.)

Hist. Bin		Count ($\times 10^3$)	% of Total
[1,3)	*All*	20.2 ± 0.59 (2.92%)	54.8% ± 1.82%
	Web	16.9 ± 0.69 (4.07%)	57.1% ± 2.68%
	Other	3.85 ± 0.049 (1.29%)	58.8% ± 0.876%
[3,7)	*All*	8.50 ± 0.59 (6.94%)	23.0% ± 1.64%
	Web	6.09 ± 0.69 (11.3%)	20.6% ± 2.37%
	Other	1.61 ± 0.049 (3.08%)	24.6% ± 0.778%
[7,15)	*All*	3.98 ± 0.59 (14.8%)	10.8% ± 1.61%
	Web	2.56 ± 0.69 (26.8%)	8.68% ± 2.33%
	Other	0.68 ± 0.049 (7.31%)	10.3% ± 0.759%
[15,31)	*All*	2.03 ± 0.59 (29.0%)	5.51% ± 1.60%
	Web	1.29 ± 0.69 (53.0%)	4.38% ± 2.33%
	Other	0.20 ± 0.049 (25.2%)	3.00% ± 0.756%
[31,63)	*All*	1.41 ± 0.59 (41.8%)	3.82% ± 1.60%
	Web	1.23 ± 0.69 (55.7%)	4.17% ± 2.33%
	Other	0.025 ± 0.049 (200%)	0.38% ± 0.755%
[63,∞)	*All*	0.946 ± 0.59 (62.3%)	2.57% ± 1.60%
	Web	0.038 ± 0.69 (1790%)	0.13% ± 2.32%
	Other	0.031 ± 0.049 (160%)	0.47% ± 0.755%

Table 9: The 10 minute mean number of inbound bytes per stream collected during periods 5, 6, and 7 (see Table 2). See also Figure 4a. (See the legend in Figure 4a or 4b for the total number of *all*, *web*, and *other* streams.)

Hist. Bin		Count ($\times 2^{20}$)	% of Total
(0,2)	*All*	155 ± 6.40 (4.11%)	42.1% ± 1.88%
	Web	99.6 ± 5.81 (5.83%)	39.6% ± 2.49%
	Other	15.9 ± 0.207 (1.30%)	54.2% ± 0.801%
[2,16)	*All*	113 ± 6.40 (5.65%)	30.7% ± 1.81%
	Web	99.2 ± 5.81 (5.85%)	39.5% ± 2.48%
	Other	3.58 ± 0.207 (5.78%)	12.2% ± 0.710%
[16,65)	*All*	39.5 ± 6.40 (16.2%)	10.7% ± 1.74%
	Web	30.0 ± 5.81 (19.4%)	11.9% ± 2.33%
	Other	0.830 ± 0.207 (24.9%)	2.83% ± 0.705%
[65,128)	*All*	13.6 ± 6.40 (47.0%)	3.69% ± 1.73%
	Web	5.51 ± 5.81 (105%)	2.19% ± 2.31%
	Other	0.186 ± 0.207 (111%)	0.632% ± 0.704%
[128,∞)	*All*	23.1 ± 6.40 (27.8%)	6.24% ± 1.74%
	Web	23.0 ± 5.81 (25.3%)	9.14% ± 2.32%
	Other	1.08 ± 0.207 (19.1%)	3.69% ± 0.705%

Table 10: The 10 minute mean number of outbound bytes per stream collected during periods 5, 6, and 7 (see Table 2). See also Figure 4b. (See the legend in Figure 4a or 4b for the total number of *all*, *web*, and *other* streams.)

Hist. Bin		Count ($\times 2^{20}$)	% of Total
(0,0.5)	*All*	174 ± 6.40 (3.68%)	47.0% ± 1.91%
	Web	97.3 ± 5.81 (5.97%)	38.7% ± 2.48%
	Other	19.4 ± 0.207 (1.07%)	66.1% ± 0.844%
[0.5,1)	*All*	111 ± 6.40 (5.75%)	30.1% ± 1.81%
	Web	94.4 ± 5.81 (6.15%)	37.6% ± 2.47%
	Other	5.19 ± 0.207 (3.99%)	17.7% ± 0.715%
[1,2)	*All*	51.2 ± 6.40 (12.5%)	13.9% ± 1.75%
	Web	37.2 ± 5.81 (15.6%)	14.8% ± 2.34%
	Other	1.70 ± 0.207 (12.2%)	5.78% ± 0.706%
[2,4)	*All*	14.4 ± 6.40 (44.4%)	3.90% ± 1.73%
	Web	13.6 ± 5.81 (42.8%)	5.40% ± 2.31%
	Other	0.412 ± 0.207 (50.3%)	1.40% ± 0.704%
[4,∞)	*All*	10.9 ± 6.40 (58.5%)	2.96% ± 1.73%
	Web	14.7 ± 5.81 (39.6%)	5.84% ± 2.31%
	Other	2.01 ± 0.207 (10.3%)	6.84% ± 0.706%

Table 11: The 10 minute mean number of total bytes per stream (inbound + outbound) collected during periods 5, 6, and 7 (see Table 2). See also Figure 12. (See the legend in Figure 12 for the total number of *all*, *web*, and *other* streams.)

Hist. Bin		Count ($\times 2^{20}$)	% of Total
(0,2)	*All*	149 ± 6.40 (4.30%)	40.3% ± 1.87%
	Web	93.4 ± 5.81 (6.21%)	37.2% ± 2.47%
	Other	21.1 ± 0.207 (0.979%)	72.0% ± 0.868%
[2,16)	*All*	134 ± 6.40 (4.76%)	36.4% ± 1.84%
	Web	97.3 ± 5.81 (5.97%)	38.7% ± 2.48%
	Other	4.84 ± 0.207 (4.27%)	16.5% ± 0.714%
[16,65)	*All*	42.3 ± 6.40 (15.1%)	11.4% ± 1.74%
	Web	38.6 ± 5.81 (15.1%)	15.3% ± 2.34%
	Other	1.86 ± 0.207 (11.1%)	6.34% ± 0.706%
[65,128)	*All*	17.3 ± 6.40 (37.0%)	4.69% ± 1.73%
	Web	14.9 ± 5.81 (39.0%)	5.92% ± 2.31%
	Other	0.404 ± 0.207 (51.2%)	1.38% ± 0.704%
[128,∞)	*All*	28.1 ± 6.40 (22.8%)	7.61% ± 1.74%
	Web	21.2 ± 5.81 (27.4%)	8.44% ± 2.32%
	Other	1.18 ± 0.207 (17.5%)	4.03% ± 0.705%