TIGER: Tor Traffic Generator for Realistic Experiments

Daniela Lopes
daniela.lopes@tecnico.ulisboa.pt
INESC-ID / IST, Universidade de Lisboa
Lisboa, Portugal

Daniel Castro
daniel.castro@tecnico.ulisboa.pt
INESC-ID / IST, Universidade de Lisboa
Lisboa, Portugal

Diogo Barradas
diogo.barradas@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Nuno Santos
nuno.m.santos@tecnico.ulisboa.pt
INESC-ID / IST, Universidade de Lisboa
Lisboa, Portugal

ABSTRACT

Tor is the most widely adopted anonymity network, helping safeguard the privacy of Internet users, including journalists and human rights activists. However, effective attacks aimed at deanonymizing Tor users' remains a significant threat. Unfortunately, evaluating the impact such attacks by collecting realistic Tor traffic without gathering real users' data poses a significant challenge.

This paper introduces TIGER (Tor traffIc GEnerator for Realistic experiments), a novel framework that automates the generation of realistic Tor traffic datasets towards improving our understanding of the robustness of Tor's privacy mechanisms. To this end, TIGER allows researchers to design large-scale testbeds and collect data on the live Tor network while responsibly avoiding the need to collect real users' traffic. We motivate the usefulness of TIGER by collecting a preliminary dataset with applicability to the evaluation of traffic confirmation attacks and defenses.

CCS CONCEPTS

• **Security and privacy → Usability in security and privacy**; **Privacy-preserving protocols**; **Pseudonymity, anonymity and untraceability**; • **Networks → Network privacy and anonymity**.

KEYWORDS

Tor; traffic analysis; dataset generation; web privacy

ACM Reference Format:
Daniela Lopes, Daniel Castro, Diogo Barradas, and Nuno Santos. 2023. TIGER: Tor Traffic Generator for Realistic Experiments. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society (WPES '23), November 26, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3603216.3624960

1 INTRODUCTION

Tor [15] is a widely used low-latency anonymity tool to evade censorship and surveillance. In a nutshell, Tor enforces anonymity by encapsulating communications in multiple layers, tunneling them through 3-relay circuits. Despite these efforts, Tor has been shown to be vulnerable to multiple traffic analysis attacks, such as website fingerprinting (WF) [14, 25, 31], traffic correlation (TC) [29, 30], and watermarking [17, 18], in which an adversary observes one or both ends of the communication to determine which client is accessing which Internet host, or to deanonymize hosts.

However, the data available to test the effectiveness of these attacks (and their mitigations) in real-world traffic is scarce, demanding the strenuous collection of datasets in conditions that may not always reflect the true characteristics of the Tor network and its users. While simulations and emulations of Tor have improved over time and become a viable option to perform experiments, capturing live Tor traffic remains paramount to evaluate the fidelity of simulators/emulators and produce accurate Tor network models [19].

Despite the fact that previous studies that attempt to simulate Tor's behavior have pointed out the difficulties involved in reproducing live experiments [21] (e.g., to consider transient conditions that might momentarily alter the network's characteristics), we argue that such perturbations are inherent to the Tor network and closer to what adversaries would experience when deploying real attacks. Unfortunately, we observe that there exists no unified framework for the automation of large-scale live experiments on the Tor network. Moreover, building such a framework with safety considerations in mind, i.e., avoiding the collection of real user data at all costs, presents multiple challenges, including the realistic mimicking of users, server endpoints, and their interactions.

To bridge this gap and simplify live Tor traffic collection without compromising real users' anonymity, we propose TIGER, a Tor traffic generator for realistic experiments, and propose guidelines to approximate the data generated by our automated clients and services to real-world Tor traffic. We also introduce a methodology to avoid overloading the Tor network, which restricts the resources used in experiments according to the network's existing capacity.

Next, after we survey the related work, we detail TIGER's architecture and how it leverages Docker containers to simplify the deployment of clients, server endpoints and onion services, and Tor relays and proxies, while ensuring portability, efficiency, and network isolation. Lastly, we run a preliminary test on TIGER for showcasing the capabilities of our framework. Specifically, we collected a large-scale dataset that can be used for the evaluation of TC attacks on Tor onion service traffic by: i) replicating multiple user profiles with the creation of sessions of diverse durations and number of requests; ii) recreating a set of representative services expected to produce diverse traffic patterns; and iii) modelling services with different access patterns.

WPES '23, November 26, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0235-8/23/11.
https://doi.org/10.1145/3603216.3624960

2 RELATED WORK

A plethora of attacks has been proposed against the Tor network [14, 17, 18, 25, 29–31, 34, 37]. However, testing the effectiveness of such attacks and defenses requires testbeds that allow for the collection of representative network datasets. Popular approaches include simulations [20, 22], emulations [4, 41], model-driven experiments [1, 3, 9, 13, 19, 42], and live experiments [7, 18, 25, 29, 31, 32].

Simulation: Simulated environments aim to accurately replicate some system metrics (e.g., bandwidth, latency, or throughput). A popular simulator for the Tor network is Shadow [20], which runs unmodified Tor software on top of a virtual network. Shadow allows replicating the Tor network using fewer resources, being leveraged to simulate relay congestion in Tor paths [2, 22] and WF attacks [21].

Emulation: Emulated environments replicate the behaviour of the Tor network by forwarding real application traffic through a set of machines that implement their own private Tor network. They allow for the assignment of realistic network conditions expected to be faced on the real Tor network, such as bandwidth and latency, and typically have a smaller scale than simulated environments since they require more resources. Many works [1, 3, 9, 13, 42] leveraged PlanetLab [11], Emulab [47], and Deter [5] to build small-scale Tor networks. However, achieving a network as representative as Tor was challenging due to limited resources and customizability. ExperimenTor [4] uses ModelNet network emulators [46] to emulate the Tor network using at least two virtual or physical machines. Shirazi et al. [40] compared Shadow and ExperimenTor and concluded that both approaches have limitations that may inaccurately model the Tor network, such as: i) Shadow ignoring relays' bandwidth weights and the geographical distribution and AS-level distribution of the Tor network, and ii) ExperimenTor having low scalability and ignoring the AS-level distribution of the network.

Models: Jansen et al. [19] proposed a Tor network model using Tor metrics [45] and used it to compare Shadow and ExperimenTor with measurements obtained from the live Tor network at a single point in time, and concluded that the decisions made in network modeling significantly influence the fidelity of testbeds. The authors found that both Shadow and ExperimenTor produce network performance and load patterns that differ from real Tor measurements.

Live experiments: Some methodologies use the live Tor network infrastructure, placing additional nodes under researchers' control for generating custom traffic to gather datasets. Several approaches have been proposed, with particular emphasis on onion services characterization [7, 32], WF attacks [10, 25, 31, 33] and defenses [24], TC attacks [29, 30] and watermarking attacks [18]. Onion services characterization generally requires controlling multiple relays to count the number of fetches for particular onion service descriptors (which is no longer possible with V3 onion services [16]), or controlling onion service guard nodes (which may also be impossible with real-world onion services since they can serve their own guard nodes). WF attacks generally require capturing traffic between the client and the client's guard node. In addition, some WF attacks [10, 25] also require recording the Tor cells exchanged on the controlled guard nodes that will be monitoring the traffic. Watermarking attacks may require using a proxy as detector of the embedded watermark or controlling a guard node [17, 18]. TC attacks require monitoring traffic at both ends of the communication.

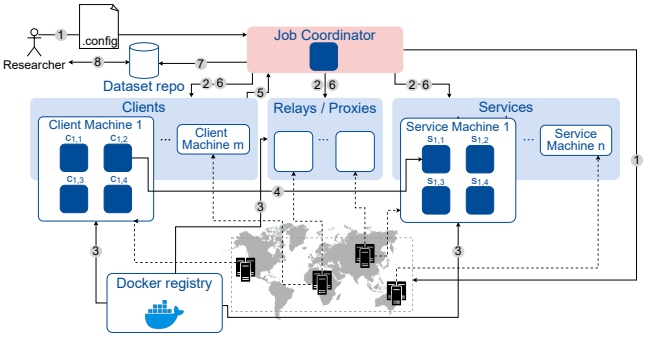


Figure 1: Design of the TIGER framework.

In summary, a framework for live experiments should allow controlling relays and proxies, as well as customizing their software, and easily modifying the functionality of clients and endpoints. It would avoid simulating numerous possible network conditions that may not accurately reflect the authentic behaviour of Tor, thereby yielding more realistic results. Furthermore, this approach provides greater flexibility in tailoring the characteristics of endpoints and is essential to test how accurate Tor network models are in simulation. Nevertheless, conducting such experiments requires extensive resources, effort, and time. Additionally, apart from the recommendations mentioned in the ethical guidelines issued by the Tor research safety board [43], live experimentation lacks standardized procedures that also consider potential abusive usage of the Tor network, such as the potential to overload the live Tor network resources. TIGER aims to address these issues by providing an experimental testbed with a configurable set of parameters that support the multiple requirements described in this paragraph.

3 DESIGN

TIGER aims to automatize the creation of datasets of Tor traffic, respecting a parameterizable set of conditions, towards approximating the traces that could be obtained from real users' traffic. Our framework aims to boost the opportunities for research with respect to attacks and defenses on Tor, while explicitly aiming to minimize additional overheads imposed to the live Tor network. TIGER does not replace simulation or emulation approaches, but it complements them by enabling easier collection of Tor real traffic to model the Tor network, and offers an alternative with particular advantages, such as for research scenarios that necessitate realistic user and service interactions with the network, such as in evaluating TC attacks, and in which we need more flexibility, such as controlling the source code (to change the Tor browser source code for instance) or controlling relays belonging to the circuit.

Challenges for setting up realistic live experiment setups: Using nodes under our control to perform live experiments requires emulating several aspects of how clients and services interact. (We refer to Internet hosts serving content simply as *services*, specifically referring to *onion services* where the distinction is required.) Specifically, we: i) model user behaviour when accessing services through Tor; ii) build representative services; iii) adjust relays and proxies to each particular scenario; and iv) create a realistic topology of a network of clients and services under our control, but that communicate via the live Tor network.

3.1 Architecture

To tackle the above challenges, we propose the framework showcased in Figure 1. TIGER leverages physical machines and containerization as the baseline for a fast, flexible, and portable deployment of a network of nodes running on top of the live Tor network. Once an experiment is over, the framework's operator is provided with a list of network captures, each encompassing the times, sizes, and directions of packets, and the IPs and ports involved. Each capture can consist of the data collected during a request, a session (i.e., a set of requests), or the full experiment, as configured by the operator.

To launch an experiment, the operator details which physical machines will be dedicated to serving clients, relays, and services, along with their characteristics if using a cloud provider, in step (1)[1]. An extra machine, *job coordinator*, orchestrates the experiment (step (2)). Each client or service machine has the ability to handle multiple client or service containers, increasing the experiment's scalability with the same resources. The container images are made publicly available in a *Docker registry* to expand TIGER's functionality according to the requirements of a given experiment. After bootstrapping the physical machines, the job coordinator proceeds to download and run the selected container images, in step (3).

Each onion service container has a key pair and a `.onion` address, and runs a Tor process for serving a pre-configurable content. Internet services expose this content but do not run any Tor-associated process. In turn, relays can run a Tor process with different possible parameters, as per Tor's Relay Guide [36]. Each client container runs a Tor client process and executes a desired crawling script to access the services and coordinate the experiment (step (4)).

An experiment ends when all clients have reached the number of requests proposed in the configuration file, by having each client send a `terminate` request to the job coordinator, as shown in step (5). Then, in step (6), the job coordinator fetches the dataset parcels from all the experiment machines, stops all the machines, and publishes the complete dataset on a repository, as in step (7). Finally, in step (8), the operator downloads the dataset from the repository.

Tor overload prevention: The Tor project provides (at least) hourly [40] relay information through CollecTor [44]. TIGER's job coordinator can periodically retrieve this data to limit the bandwidth exerted by each client on the Tor network, ensuring the experiment being run does not straggle Tor's network capacity.

Prototype: We developed a TIGER protoype using the `terraform` tool as an orchestrator to launch physical machines, Google Cloud as the cloud provider, `ansible` as the provision tool to set up the machines and deploy the containers, and *Docker Hub* to share the container images. We created our own images for the job coordinator, and all clients and services. To preserve ground truth about the start and end times at which each request/session took place (i.e., to synchronize traffic captures collected by the clients and services involved in communication), we rely on REST endpoints — contacted by placing requests outside the Tor network – to allow clients to flag the services they're communicating with about the start/finish of each request or session, shown in Appendix A.1.

[1]These are specified in a configuration file, which may include: the number of clients, which machines will act as generic Internet services, onion services, or relays; the zones to run machines in case of a cloud provider, or the IP addresses of machines belonging to clusters; the container image IDs to use; the generic services and onion services to serve; and the guard nodes that clients and onion services will make use of.

3.2 Setting up Realistic Live Experiments

Emulating clients: Emulating the usual behavior of Tor clients is a little-explored topic due to the lack of studies on real Tor users' access patterns. Thus, we base our recommendations on Internet usage studies [12, 26, 27, 39] and studies on the effectiveness of WF attacks on real data [23], which suggest modeling the following characteristics: i) stay times in pages, which has been shown to follow a Weibull distribution [26] and depend on the content of the website [39]; ii) visiting inner pages of each website [23]; iii) considering the probability of users to start a new session (e.g., moving on to a different website) [27]; iv) simulating multi-tab browsing [12, 23]; v) re-visiting rate [12]; vi) usage of multiple Tor browser versions [23]; and vii) mimicking different browsing patterns [12]. All these combined allow us to devise a realistic crawling methodology to control the accesses performed by each client to different services, allowing us to produce multiple variations, e.g., each implemented by a different client Docker container image.

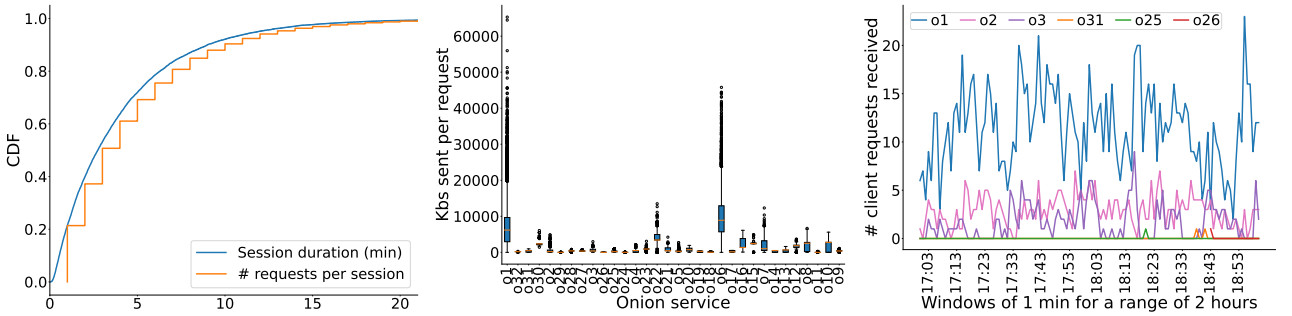
Emulating services: In the past, Mani et al. [28] controlled multiple exit relays to study the main clearnet destinations of Tor users', finding that most Tor users' typically access websites that overlap the most popular websites visited by non-Tor users. Following these findings, TIGER operators may either choose to mirror a set of top-visited websites (e.g., included in the Tranco list [35]), or simply access popular Internet websites via TIGER-controlled proxies.

The realistic emulation of accesses to onion services, however, presents additional challenges. For an accurate TIGER live experiment, onion services should experience similar access patterns and serve similar contents as those observed in real onion services. However, existing studies on onion services are outdated and new studies currently limited due to the recent privacy improvements on onion services [16] (which, e.g., no longer allow matching the *Hidden Service Directory* descriptor fetches to the onion services).

Still, previous studies [6, 32] on the ecosystem of existing onion services found that most accesses by Tor clients are targeted towards a relatively small set of popular onion services, which follows a similar trend to that observed with typical Internet services [8]. These can be modeled in TIGER as a Zipf distribution, where the most popular website is accessed significantly more often than others, while the least accessed websites receive very few accesses.

Finally, TIGER requires the mirroring of a set of representative web services being served by onion services. Fortunately, previous studies [6, 32] on the onion service ecosystem have characterized prevalent categories of content served by such services. A possible way to integrate these representative services within TIGER is to manually collect onion services bound to each of these categories by searching for these topics on the `ahmia.fi` onion service search engine. We note that onion services serving content other than HTTP and HTTPS (such as those primarily used for SSH or Bitcoin nodes) may also be added manually to TIGER-controlled nodes.

Emulating proxies: For experimenting with certain classes of attacks on Tor, it is essential to deploy proxy nodes that capture the traffic exchanged with network entities outside of the researchers' control (e.g., TC [29, 30] or certain WF [10] attacks where clients access real-world websites via exit relays that are not under the researchers' control). In addition, Rimmer et al. [38] found that, in



(a) Session durations and requests per session CDF. (b) Data sent by request per onion service. (c) Different access patterns by onion service popularity.

Figure 2: Generated dataset statistics.

the settings above, employing a single proxy for live traffic collection may alter the characteristics of traffic and skew the conclusions about an attack's effectiveness. To address this, Rimmer et al. propose deploying multiple proxies in the traffic collection infrastructure. As described in Section 3.1, TIGER can easily support the deployment of multiple geographically-distributed proxies on the configuration of an experimental run.

Emulating topology: Juarez et al. [23] have showed that the performance of WF attacks is degraded when a classifier is trained in a geographical location which is different than the one where the classifier will be used as part of a WF attack. This may also be the case for adversaries with limited geographical coverage and that aim to launch TC attacks [29]. Thus, we posit that researchers might have interest to mimic Tor nodes' geographical distribution when creating large-scale datasets resorting to live-experiments. TIGER allows researchers to flexibly host their clients and services across a range of machines located worldwide, including cloud machines in different zones, or machines provided in the scope of collaboration with other research institutes.

4 RESULTS

This section describes a TIGER-generated dataset designed for supporting the evaluation of TC-style attacks targetting onion services. The dataset was crafted to serve as a versatile example, requiring a distributed infrastructure to capture traffic at various locations.

TIGER setup: The corresponding TIGER setup is as follows. The operator configures 10 machines to host clients and 8 machines to host onion services. Each machine serves 4 instances of Tor clients and onion services, respectively. Following TIGER's Docker support, we created a container image for all clients, and another for all onion services, containing 32 different websites mirrored from real-world onion services. The experiment is configured so that each client performs 2500 requests, also uniformly accessing clearnet services listed in Tranco's list [35] in-between sessions established towards onion services. Following a Zipf distribution, onion services' access rate probability ranged between 44.2% (`onion1`) and 0.2% (`onion26`). The most popular onion services were evenly distributed across physical machines so as not to overload a single machine with the majority of traffic exchanged during the experiment. We configured clients to have 80% probability of staying on the same website and 20% probability of starting a new session. Each access is ruled by a variable stay time of up to 10 minutes.

Dataset characteristics: We focus on describing how the obtained dataset comprises realistic user behaviors via Tor. We note that an operator may configure TIGER to fine-tune the characteristics of the generated traces for considering different scenarios.

Figure 2(a) shows that sessions (sequences of requests to the same service) display highly variable duration and request count, effectively simulating distinct user browsing patterns, e.g, being useful to evaluate how different browsing patterns affect the accuracy of traffic analysis attacks. Figure 2(b) shows the high variability of the onion services' characteristics that would translate into different patterns when accessed (for instance, `onion1` is expected to send higher volumes of traffic per request), e.g, being useful to evaluate the properties that can make a service's traffic more easily recognizable. Figure 2(c) shows the concurrent client requests received by the three most and least popular onion services per minute, for a duration of 2 hours. We observe that the TIGER experiment allows for producing concurrent traffic aimed at onion services, which might be useful for assessing the effectiveness of traffic analysis attacks and defenses under different noisy conditions. We provide complementary results in Appendix A.2.

Limitations: TIGER's main limitations pertain to the scalability of the experimentation, which necessitates the use of multiple machines and a considerable time to generate a dataset containing numerous sessions. However, once the dataset is successfully compiled, it can be widely applied, provided that all configurations align with the specific requirements of the intended experiment.

5 CONCLUSIONS AND FUTURE WORK

This paper introduces TIGER, a versatile framework that generates realistic Tor traffic datasets through live network experiments, while preserving user privacy and network resources.We used TIGER to generate an example dataset that can be used for traffic correlation purposes. TIGER's tunability allows operators to tailor experiments to their specific needs, making it a powerful tool for studying different traffic analysis attacks on Tor.

Future directions to improve TIGER include: i) build client images that can access private pages (i.e., bypass login screens), or filling CAPTCHAs; ii) devise a tool that can validate Tor network simulators using TIGER datasets; iii) validate the generated datasets through testing with established TC and WF attacks; and iv) perform a comparative analysis against datasets generated using Shadow.

REFERENCES

[1] Masoud Akhoondi, Curtis Yu, and Harsha V. Madhyastha. 2012. LASTor: A Low-Latency AS-Aware Tor Client. In *IEEE Security and Privacy*.

[2] Armon Barton, Matthew Wright, Jiang Ming, and Mohsen Imani. 2018. Towards Predicting Efficient and Anonymous Tor Circuits. In *USENIX Security*.

[3] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. 2007. Low-Resource Routing Attacks against Tor. In *ACM WPES*.

[4] Kevin Bauer, Micah Sherr, and Dirk Grunwald. 2011. ExperimenTor: A Testbed for Safe and Realistic Tor Experimentation. In *CSET*.

[5] Terry Benzel. 2011. The Science of Cyber Security Experimentation: The DETER Project. In *ACM ACSAC*.

[6] Alex Biryukov, Ivan Pustogarov, Fabrice Thill, and Ralf-Philipp Weinmann. 2014. Content and Popularity Analysis of Tor Hidden Services. In *IEEE ICDCS Workshops*.

[7] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. 2013. Trawling for tor hidden services: Detection, measurement, deanonymization. In *IEEE Security and Privacy*.

[8] L. Breslau, Pei Cao, Li Fan, G. Phillips, and S. Shenker. 1999. Web caching and Zipf-like distributions: evidence and implications. In *IEEE INFOCOM*.

[9] Sambuddho Chakravarty, Marco V. Barbera, Georgios Portokalidis, Michalis Polychronakis, and Angelos D. Keromytis. 2014. On the Effectiveness of Traffic Analysis against Anonymity Networks Using Flow Records. In *Passive and Active Measurement*.

[10] Giovanni Cherubin, Rob Jansen, and Carmela Troncoso. 2022. Online Website Fingerprinting: Evaluating Website Fingerprinting Attacks on Tor in the Real World. In *USENIX Security*.

[11] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. 2003. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communications Review* (2003).

[12] Kyle Crichton, Nicolas Christin, and Lorrie Cranor. 2022. How Do Home Computer Users Browse the Web? *ACM Transactions on the Web* (2022).

[13] Anupam Das and Nikita Borisov. 2013. Securing Anonymous Communication Channels under the Selective DoS Attack. In *Financial Cryptography*.

[14] Xinhao Deng, Qilei Yin, Zhuotao Liu, Qi Li, Mingwei Xu, Ke Xu, and Wu Jianping. 2023. Robust Multi-tab Website Fingerprinting Attacks in the Wild. In *IEEE Security and Privacy*.

[15] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security*.

[16] Tobias Hoeller, Michael Roland, and René Mayrhofer. 2021. On the State of V3 Onion Services. In *ACM FOCI*.

[17] Alfonso Iacovazzi, Daniel Frassinelli, and Yuval Elovici. 2019. The DUSTER attack: Tor onion service attribution based on flow watermarking with track hiding. In *ACM RAID*.

[18] Alfonso Iacovazzi, Sanat Sarda, and Yuval Elovici. 2018. Inflow: Inverse Network Flow Watermarking for Detecting Hidden Servers. In *IEEE INFOCOM*.

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

[20] Rob Jansen and Nicholas Hopper. 2011. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *NDSS*.

[21] Rob Jansen and Ryan Wails. 2023. Data-Explainable Website Fingerprinting with Network Simulation. (2023). See also https://explainwf-popets2023.github.io.

[22] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. 2013. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In *ACM CCS*.

[23] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. 2014. A Critical Evaluation of Website Fingerprinting Attacks. In *ACM CCS*.

[24] Marc Juárez, Mohsen Imani, Mike Perry, Claudia Díaz, and Matthew Wright. 2015. WTF-PAD: Toward an Efficient Website Fingerprinting Defense for Tor. (2015). http://arxiv.org/abs/1512.00524

[25] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. 2015. Circuit fingerprinting attacks: Passive deanonymization of tor hidden services. In *USENIX Security*.

[26] Chao Liu, Ryen W. White, and Susan Dumais. 2010. Understanding Web Browsing Behaviors through Weibull Analysis of Dwell Time. In *ACM SIGIR*.

[27] Anna Harbluk Lorimer, Lindsey Tulloch, Cecylia Bocovich, and Ian Goldberg. 2021. OUStralopithecus: Overt User Simulation for Censorship Circumvention. In *ACM WPES*.

[28] Akshaya Mani, T. Wilson-Brown, Rob Jansen, Aaron Johnson, and Micah Sherr. 2018. Understanding Tor Usage with Privacy-Preserving Measurement. In *ACM IMC*.

[29] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. 2018. DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning. In *ACM CCS*.

[30] Se Eun Oh, Taiji Yang, Nate Mathews, James K Holland, Mohammad Saidur Rahman, Nicholas Hopper, and Matthew Wright. 2022. DeepCoFFEA: Improved flow correlation attacks on Tor via metric learning and amplification. In *IEEE Security and Privacy*.

[31] Rebekah Overdorf, Mark Juarez, Gunes Acar, Rachel Greenstadt, and Claudia Diaz. 2017. How unique is your .onion?: An analysis of the fingerprintability of tor onion services. In *ACM CCS*.

[32] Gareth Owen and Nick Savage. 2016. Empirical analysis of Tor hidden services. In *IET Information Security*.

[33] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. 2016. Website Fingerprinting at Internet Scale. In *NDSS*.

[34] Andriy Panchenko, Asya Mitseva, Martin Henze, Fabian Lanze, Klaus Wehrle, and Thomas Engel. 2017. Analysis of Fingerprinting Techniques for Tor Hidden Services. In *ACM WPES*.

[35] Victor Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczynski, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation.

[36] The Tor Project. 2020. TorRelayGuide. https://gitlab.torproject.org/legacy/trac/-/wikis/TorRelayGuide#Parttwo:technicalsetup. Accessed: 2023-07-15.

[37] Mohammad Saidur Rahman, Payap Sirinam, Nate Mathews, Kantha Gangadhara, and Matthew Wright. 2020. Tik-Tok: The Utility of Packet Timing in Website Fingerprinting Attacks. In *PoPETS*.

[38] Vera Rimmer, Theodor Schnitzler, Tom Van Goethem, Abel Romero, Wouter Joosen, and Katharina Kohls. 2022. Trace Oddity: Methodologies for Data-Driven Traffic Analysis on Tor. In *PoPETS*.

[39] Kimberly Ruth, Aurore Fass, Jonathan Azose, Mark Pearson, Emma Thomas, Caitlin Sadowski, and Zakir Durumeric. 2022. A World Wide View of Browsing the World Wide Web. In *ACM IMC*.

[40] Fatemeh Shirazi, Matthias Goehring, and Claudia Diaz. 2015. Tor Experimentation Tools. In *IEEE Security and Privacy Workshops*.

[41] Sukhbir Singh. 2014. Large-scale emulation of anonymous communication networks. In *Master's thesis, University of Waterloo*.

[42] Can Tang and Ian Goldberg. 2010. An Improved Algorithm for Tor Circuit Scheduling. In *ACM CCS*.

[43] The Tor Project. 2016. Research Safety Board. https://research.torproject.org/safetyboard/. Accessed: 2023-06-04.

[44] The Tor Project. [n. d.]. CollecTor. https://metrics.torproject.org/collector.html. Accessed: 2023-06-04.

[45] The Tor Project. [n. d.]. Tor Metrics. https://metrics.torproject.org/. Accessed: 2023-06-04.

[46] Kashi Venkatesh Vishwanath, Diwaker Gupta, Amin Vahdat, and Ken Yocum. 2009. ModelNet: Towards a datacenter emulation environment. In *IEEE Conference on Peer-to-Peer Computing*.

[47] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. 2003. An Integrated Experimental Environment for Distributed Systems and Networks. In *ACM SIGOPS Oper. Syst. Rev.*

A APPENDIX

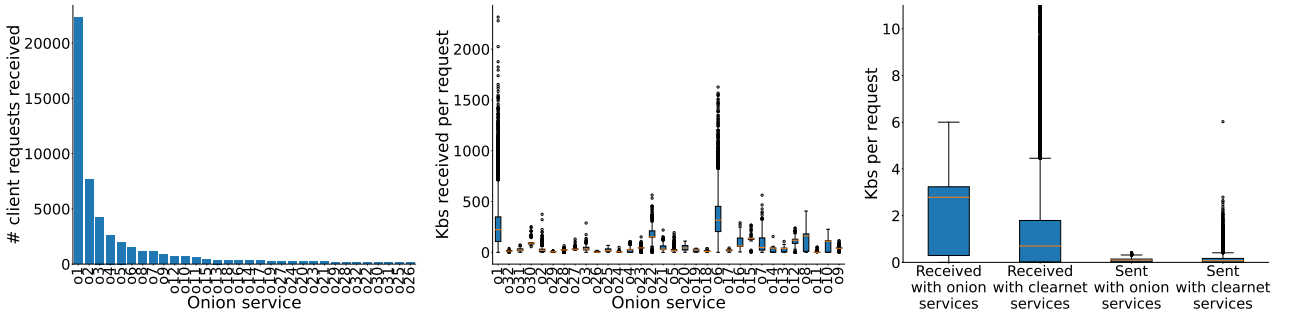
A.1 Clients' and services' example code

Algorithm 1 gives a brief demonstration of the HTTP server for client coordination being executed by each service in parallel with the service to be accessed through Tor. In the case of a Tor node, such as an onion service or a relay, that service should be associated with a Tor process configured correctly with a torrc file.

```
# Create endpoint /startTrafficCapture
def start_traffic_capture():
    # Start traffic capture based on requests' content, such as being a session
    ↪ capture, or a request capture, all controlled by the client
    cmd = f'tcpdump -i any -s 66 -W 1 -w pcap_file.pcap'
    os.system(f'docker exec onion-img-{id} sh -c "{cmd}"')
# Create endpoint /stopTrafficCapture
def stop_traffic_capture():
    # Stop an existing traffic capture to end a request or a session
    os.system(f'docker exec onion-img-{id} sh -c "kill {pcap_pid}"')
# Serve an HTTP server on REST_PORT to receive traffic capture commands
...
```

Algorithm 1: Onion service Python code snippet

Algorithm 2 describes a general skeleton to implement the desired client browsing behavior. We modeled the characteristics i) and ii) in Section 3.2 and implemented a different configuration to visit clearnet services and onion services, keeping it extensible to add further functionality that implements different client browsing behaviors. We highlighted in magenta the tunable parameters.



(a) Onion service popularity expressed by the number of requests made by clients.

(b) Data received by request per onion service.

(c) Data received and sent by clients when accessing onion services and clearnet services.

Figure 3: Additional generated dataset statistics.

```python
class ServiceHandler:
    # Common code for each service
    ...
class SeleniumTorBrowserHandler(ServiceHandler):
    # Encapsulates the browser functionality, in this case, Selenium's Tor browser
    driver: TorBrowserDriver
    # Initializations and start Selenium Driver
    ...
    def visit_new_website(self, url: str):
        self.driver = start_selenium_driver(self.driver)
        self.driver.get(url)
        return self.driver.page_source
    def visit_random_webpage(self):
        # Find all the links referenced within the HTML of the current page the
        ↪ user is on and add to the list of past visitable pages found within
        ↪ the same session
        ...
        # Choose a random visitable page and access the corresponding URL using
        ↪ the Selenium Driver
        ...
class SessionHandler:
    # Models the client's behavior during each session
    browser: SeleniumTorBrowserHandler
    actions = Dict[str, float]
    cap_folder: str
    sample_name: str
    session_capture_cmd: str
    # Initializations and start browser handler
    self.actions = {'new_site': NEW_SITE_PROB, 'navigate': SAME_SITE_PROB}
    ...
    def choose_next_action(self):
        return np.random.choice(len(self.actions), 1,
        ↪ p=list(self.actions.values()))
    def start(self, client_id):
        self.session_capture_cmd = start_traffic_capture(...)
    def visit_new_website(self, url: str):
        # visits a different site (given in url)
    def visit_random_webpage(self):
        # follows a random link within the current webpage
class SessionHandlerToOnions(SessionHandler):
    onion_nodes: dict
    hostname: str
    # Initializations and different implementations to work with onion services
    ...
if __name__ == '__main__':
    stay_times = np.random.weibull(a=SHAPE, size=request_iters} * SCALE # Wait
    ↪ between requests
    # More initializations to maintain state
    ...
    request_counter = request_iters
    full_client_capture = start_traffic_capture(...)
    while request_counter > 0:
        # Update counters
        ...
        clearweb = random.randint(0, 100) < clearnet_probability*100
        # Choose random clearweb or onion service to visit
        base_url = ...
        session = SessionHandler(...) if clearweb else SessionHandlerToOSes(...)
        session.start()
        session.visit_new_website(base_url)
        next_action = session.choose_next_action()
        stay_time = stay_times[request_id]
        sleep(stay_time)
        while next_action == 'navigate':
            # Update counters
            ...
            next_action = session.choose_next_action()
            session.visit_random_webpage(...)
            sleep(stay_times[request_id])
        # Update session_id at the end of each session
        ...
stop_traffic_capture(full_client_capture, coordinator)
```

Algorithm 2: Client Python code snippet

A.2 Extra Generated Dataset Statistics

We show further statistics on the generated dataset. Figure 3(a) shows the disparities in onion service popularity, as expected when using a Zipf distribution. `onion1` received more than half of all requests, with the majority receiving few requests. This enables analyzing important characteristics mostly overlooked in previous work, such as stream separation at the onion service side, performance deterioration of WF, TC, and watermarking attacks when services handle multiple requests simultaneously, Tor network performance analysis, among others.

Figure 3(b) shows the high variability of the onion services' characteristics that would translate into different patterns when receiving client requests, showing that incoming traffic follows a similar distribution to outgoing traffic based on the served pages' volumetric characteristics, but with much lower traffic volumes when compared to Figure 2(b).

Figure 3(c) shows the different traffic volume characteristics when clients are accessing onion services and clearnet services through Tor. This allows, for instance, to evaluate classifiers that distinguish Tor traffic to Internet services from Tor traffic to onion services, or to analyze the feature importance of directional packet volumes, or to analyze how onion service's characteristics can affect the results of attacks or countermeasures evaluated on Tor traffic to Internet services.

The evaluation presented shows how TIGER can be tuned to produce datasets with highly variable characteristics, depending on the intended application.

B ACKNOWLEDGMENTS

We thank the anonymous reviewers for their comments and insightful feedback. This work was supported by the Fundação para a Ciência e Tecnologia (FCT) under grants PRT/BD/154197/2022 and UIDB/50021/2020, by NSERC under grant RGPIN-2023-03304, and by IAPMEI under grant C6632206063-00466847 (SmartRetail).