RLBrowse: Generating Realistic Packet Traces with Reinforcement Learning

Alexander Griessel*, Maximilian Stephan*, Martin Mieth†, Wolfgang Kellerer*, Patrick Krämer*
*Technical University of Munich, Munich, Germany
†ipoque GmbH - A Rohde&Schwarz Company, Leipzig, Germany

***Abstract*—Automated Web Browsing tools, such as `Selenium` and headless browsers, are used to collect traffic traces from networked applications, with which statistical models describing the traffic are obtained. However, we show that traces from Selenium and headless browsers have markedly different traffic characteristics than human generated traces, with potential impact on the quality of the obtained models. To overcome this limitation, we propose RLBROWSE, an automated web automation framework that imitates human browsing habits by separating web automation from the browser using reinforcement learning. By separating the browser and automation tool, RLBROWSE improves on 9 out of the 13 traffic trace features tested. The distribution of packet sizes in a trace improves the most, with a nearly $400\,\%$ improvement. We test RLBROWSE by collecting a corpus of network packet traces on a set of human-navigated website browsing sessions, and by RLBROWSE and `Selenium`. In the subsequent analysis, we identify key differences in the resulting packet traces.**

I. INTRODUCTION

In recent years, the number of published attacks, exploits and vulnerabilities in network traffic has led to a state of encrypt-everything [1]. According to the Google Transparency Report, the share of pages loaded using `HTTPS` (Hypertext Transfer Protocol Secure) has increased from between 30 % and 47 % in 2015, to between 80 % and 98 % today. While the move towards encryption protects individual security [2], and prevents network censorship [3], [4], encrypted network traffic renders network monitoring difficult [1]. With traffic encryption, statistical traffic classification through traditional techniques, like Deep Packet Inspection (DPI) is no longer possible. However, classification of network traffic into network applications, and network traffic monitoring is used by Internet Service Providers for QoS guarantees, detection of malicious intrusions, and network traffic management [4]–[6]. For encrypted traffic, Internet Service Providers and businesses interested in their own network management need to look towards new traffic classification methods, such as machine-learned models that make use of exposed packet-level features-packet timing, sizes, and direction [7].

Machine-learning solutions require high-quality, large-scale, representative data of traffic traces of networked applications. To date, the network traffic these solutions are trained on are generated by automated tools [8]–[11]. But, to the best of our knowledge, it has not yet been investigated how representative synthetically generated traffic is for human generated traffic. We fill this gap and compare the traffic generated with Selenium and headless browsers against human generated traces. We limit the evaluation to website interaction, since website classification has a magnitude more classes than the more generally defined traffic classification, making the gathering of large data-sets difficult, thus requiring automated methods [7], [12]–[14]. We show that traffic characteristics differ significantly for synthetic and manual traces. We determine that RLBROWSE improves on 9 out of the 13 traffic trace features tested. Further, we find that several features, including the distribution of packet sizes, are improved by up to 400%.

To generate more realistic traffic, we propose RLBROWSE, a framework that mitigates trace differences observed with `Selenium`. Unlike existing tools, RLBROWSE is built on the foundation that human-representative web automation requires a complete separation of browser and automation tool. Compared to `Selenium` and `Selenium Headless`, RLBROWSE improves the divergence in packet size distribution by up to $347\,\%$. Average packet sizes are improved from a $26\,\%$ difference to $2\,\%$. Further, significant improvements are seen in outgoing burst rate features. In fact, RLBROWSE generates traffic with nearly identical packet sizes, trace lengths, and outgoing bursts as human generated traffic. Further, because RLBROWSE navigates web-pages by emulating human browsing, any differences in traces generated by the two methods are independent from browser configuration or web-page interaction methodology. This ensures that RLBROWSE is unaffected by browser updates, and that website interaction is congruent regardless of the browser used. Further, RLBROWSE is not affected by closures or restrictions in access to an API or `HTML` identifiers.

Our contributions are as follows:

- We demonstrate that significant differences exist between traces generated through existing web automation tools, and traces generated through human browsing.
- We propose RLBROWSE, an innovative tool for automated generation of traffic traces in which a Reinforcement Learning trained agent navigates the web-page using only mouse and keyboard.
- We evaluate that traces generated by RLBROWSE are representative of human generated traces, in the websites investigated in our dataset.
- We show that a minimal implementation of RLBROWSE performs better in respect to trace similarity, than the existing tools `Selenium`, and `Selenium Headless`.

978-1-6654-0601-7/22/$31.00 © 2022 IEEE

The remainder of this paper is structured as follows: Section II elaborates on related work and background. Section III introduces and discusses the design goals, framework, and implementation of RLBROWSE. The comparison dataset, data collection, and feature selection are elaborated in Section IV. Further, Section IV presents the results and findings drawn from the evaluation of our comparison dataset. Finally, Section V concludes this paper with a summary of key findings and a short discussion of the future work.

II. BACKGROUND AND RELATED WORK

A. Web Automation

Web automation tools are a pair of browser, and automation tool. A number of browsers have been used in network tracing applications. These include headless browsers such as `PhantomJs` [15], `Firefox Headless` and `Chromium Headless` [16]. The most popular automation tool is `Selenium` [8], [10]. `Selenium` utilizes `DOM` (Document Object Module) nodes to help identify web elements on a web-page. `DOM` identifiers are often modified between web-page reloads, preventing automation scripts from reliable playback of a pre-programmed sequence [17]. During implementation of the `Selenium` automation scripts for this paper, `DOM` identifiers such as *ids* and *class* changed consistently. RINGER [17] extends `Selenium` by logging as many identifiers on a `DOM` element as possible. While this makes RINGER more robust against changes to a web-page and against identifier obfuscation, it does not give a guarantee of success, with automation tests succeeding only 74 % of the time, and decaying by 7 % over a three week period. COCO [18] is a conversational web automation tool. Given a short command by a user, COCO generates a plan to reach the requested information, after which it executes the plan on the web, and extracts and returns results to the user. COCO uses JavaScript to browse the web with `Firefox` browsers. Finally, COSCRIPTER is a similar tool to RINGER, and navigates across the web using `DOM` attributes [19]. RLBROWSE differs from these tools in that it does not navigate websites by `DOM`, but rather by mouse and keyboard. Further, RLBROWSE remains independent from the browser at all steps, limited to the same input a human user would receive. Thus, RLBROWSE does not require specific browser configurations or APIs.

B. Crawler Identification

A web crawler is a scripted bot that navigates across the web, using a web automation tool. As discussed, most web automation tools interface directly with a browser. Vastel et al. and Chen et al. [16], [20] present comprehensive studies on web-crawlers, and their detection. Vastel et al. [16] found that crawlers are detected using browser or automation framework specific attributes properties- for example, `Selenium` injects additional properties into the `htmldom document` class: `__webdriver_script_fm`, `__fxdriver _unwrapped` among others. In another case, images were fed to the requesting browser, and the size of the placeholder image read, as several headless browsers returned 0x0 Pixel placeholder images. Since web-crawlers are built with the same tools used to automate packet trace collection, the differences that identification techniques exploit are likely to have an impact on traffic generated by these tools. In contrast, RLBROWSE is separated from the browser. Thus, browser configuration and resource requests are independent from RLBROWSE interacting with a web-page.

III. RLBROWSE: AUTOMATED WEB BROWSING

In this paper, we define web browsing over a single website and the web-pages the website encompasses. Web browsing is a task that begins from a user's current position on a given web-page, and extends to the *browsing objective*, or the end goal of the user, on the same website. For example, a user might begin at *google.com*, and have the objective to look at cat pictures. In this case, the user's browsing objective would be the *google.com* image results web-page containing pictures of cats. Web browsing is a sequential task [21], [22]- even when a user desires to accomplish multiple tasks simultaneously, the user is limited by control over a single mouse cursor. Thus, an arbitrary web browsing objective can be broken down into a series of navigation steps, or interactions. Interactions are composed of a navigation component, where an agent navigates to a given destination d_i, and condition c_i to begin an action a_i. In the previous example, an agent navigates the mouse cursor to the *google.com* search bar, and when the condition *"mouse cursor state changes to clickable"* occurs, the agent executes the action *"click"*, proceeding to enter the word *"cats"*. Cursor state changes can be identified independently from a browser event, either by a visual change in the cursor, or by reading the cursor state from the operating system. This structure offers two opportunities for which reinforcement learning could provide a benefit. First, the structure of a task implicates a planned sequence of actions. A reinforcement learning agent can improve on existing web automation tools by planning actions to reach a browsing objective. Second, mouse control is a necessity for the separation of browser and automation tool. The movement of a mouse is a known stimulus for robot checks and CAPTCHA pop-ups [23], [24].

A. Design Goals

The RLBROWSE framework should fulfill the following design goals:

- RLBROWSE should only interface with a web browser by means of mouse cursor or keyboard, such that RLBROWSE interacts identically with any given browser, and does not change browser settings.
- RLBROWSE should only execute a navigational step if the prior navigational step is completed: when the condition c_i is met, and action a_i executed. Browsing behavior should be sequential, and actions not completed in a manner not possible with a mouse cursor and keyboard.
- RLBROWSE should respect navigation timing: Actions should not be completed instantaneously, nor should they preempt cues a user would realistically require to complete the action.

- RLBROWSE should not attempt a navigation to a destination d_i if the web content at d_i is not loaded. This condition prevents pages from partially loading into a state a user would not normally be capable of interacting with, before the page load is stopped due to a new navigation.

B. Framework

The RLBROWSE framework is a web automation tool built on reinforcement learning, and composed of four major components: the INPUT MODULE, ACTION ELEMENTS, the AGENT, and the ACTION MODULE. RLBROWSE structures web browsing as chained tasks. The INPUT MODULE represents the observational interface available to the AGENT. In this paper, RLBROWSE is implemented with the low-level observation space S_1, which is composed of the current and previous cursor positions: (x^t, y^t), (x^{t-1}, y^{t-1}), current and previous velocity: (v_x^t, v_y^t), (v_x^{t-1}, v_y^{t-1}), as well as the predicted next position, given the current velocity: (x^{t+1}, y^{t+1}). Finally, the target position (X, Y) is also given, where the target position is given by the current ACTION ELEMENT. Future work includes an extension in which the agent navigates via screen input, and identifies web-elements visually [25]. RLBROWSE trains the AGENT with Soft-Actor-Critic (SAC). SAC is well-suited for environments with multi-modal strategies and continuous action spaces, as in our case [26]. The ACTION MODULE implements the action space for the agent. The agent interacts with a web-page in two ways. First, the AGENT has control over the velocity of the mouse cursor, (v_x^t, v_y^t). Second, the AGENT controls keyboard commands, such as typing or pressing the *"enter"* key. In this paper, we limit the implementation of the second action space to scripted events executed by an ACTION ELEMENT.

The AGENT is defined as any algorithmic or learned method that converts the input I from the INPUT MODULE into an action a^i, or a navigational step (v_x^t, v_y^t), in the ACTION MODULE, conditioned on the current ACTION ELEMENT. More generally, an AGENT utilizes the input from the framework to produce an output action such that the AGENT achieves the goal defined in the ACTION ELEMENT. AGENTS are non-invasive, meaning that they are limited to browser interactions a regular web user would make. AGENTS cannot find page elements by interpreting the `HTML` page source, and cannot interface with the web browser through APIs. RLBROWSE uses reinforcement learning to browse a website, by navigating over its web-pages. We define the reward function for the mouse navigation goal as $R = -\alpha\|p^t - p^{t-1}\| - \beta t$, where p^t is the current mouse position, p^{t-1} the previous mouse position, and α the scalar factor for the position-based reward. Additionally, the agent receives a negative reward for increasing time steps.

Each ACTION ELEMENT represents a single interaction with the web browser, containing the specific action the AGENT should take- such as the navigation destination, the action the AGENT should take when reaching the destination, and what constitutes the end of the action. Once an ACTION ELEMENT has reached the end of its action, the next ACTION ELEMENT is loaded and given to the AGENT. AGENTS are interchangeable, and independent of the ACTION ELEMENT. They are responsible for the task of navigation: controlling a mouse or keyboard to reach an on-screen destination. The AGENT receives its directive from the ACTION ELEMENT, and navigation-necessary input from the INPUT MODULE.

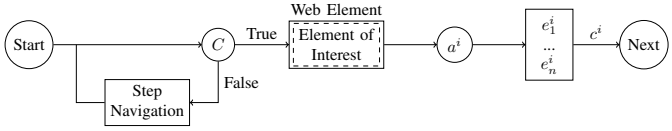


Fig. 1: Flow Diagram of Steps within an Action Element.

ACTION ELEMENTS are either predefined, or defined dynamically when RLBROWSE determines a new objective. ACTION ELEMENTS are composed of three blocks of information:

1) **Element of Interest** Defined as a bounding box containing a region that an agent should navigate to. Elements of Interest can be generated dynamically, given an agent's observation, or can be predefined in minimal implementations of RLBROWSE. In this paper, the Element of Interest is used to generate the target position (X, Y).
2) **Condition** The condition C is responsible for executing the Action when a condition is met. For example, an agent may need to scroll if it cannot reach the Element of Interest without doing so. Or, an agent attempts to click on the element, but only if within the bounding box of the Element of Interest.
3) **Action** A tuple containing an action a^i, action events e^i, and action completion c^i: $A^i = (a^i, e^i, c^i)$. The action initiates a number of non-navigational events, which are completed with the final event, the completion. For example: a^i = "click", e^i = (*"type text"*, press *"Enter"* key), and c^i = *"wait"*.

An ACTION ELEMENT represents an AGENT's current goal for the given navigation. ACTION ELEMENTS are designed to be chained together, to fulfill the design goal of a sequential browsing model. The individual parts of the ACTION ELEMENT and the corresponding steps are shown in Figure 1. ACTION ELEMENTS begin at some starting point, from which they navigate in steps towards the Element of Interest. Once the condition C is met, the AGENT is considered to have reached the Element of Interest, and initiates the action a^i, completing the series of action events e^i. Once the completion condition c^i is met, the next ACTION ELEMENT begins.

IV. EVALUATION

A. Dataset

Network packet traces were gathered for four methods of web interaction and five websites. Each website was assigned a trial consisting of a number of defined ACTION ELEMENTS. For each of the four methods of web interaction, the browsing objective and the individual navigation steps are identical. Trials begin on the index page of a website, and proceed over two website sub-pages. Navigation between consecutive pages is performed by accessing a linking web element, e.g. a button,

TABLE I: Features and Corresponding Values. Sel. is short for `Selenium`, Headl. for `Headless`, and RLB. for RLBROWSE.

Feature	Unit	Sel.	Headl.	RLB	Human
Packet Size Distribution	$d_{measure}$	0.242	0.219	**0.063***	N/A
Packets Inbound Distribution	$d_{measure}$	0.27	**0.22**	0.33	N/A
Packets Outbound Distribution	$d_{measure}$	0.28	**0.26**	0.31	N/A
Avg. Packet Size	Bytes	667	661	**542**	531
Var. Packet Size	KBytes^2	0.433	0.403	**0.336**	0.331
Avg. Burst Packets Incoming	Unitless	2.76	**2.89**	2.73	4.22
Avg. Burst Packets Outgoing	Unitless	7.01	10.40	**3.94**	3.68
Avg. Burst Bytes Incoming	Bytes	8930	**10559**	9148	10907
Avg. Burst Bytes Outgoing	Bytes	4510	7351	**1468**	1708
No. of Packets Inbound	Unitless	475	281	**571**	527
No. of Packets Outbound	Unitless	588	704	**976**	1029
Avg. Interarrival Time	ms	4.7	6.5	**5.6**	5.8
Var. Interarrival Time	s^2	2.14	6.24	**2.44**	3.96

*(Bold) Synthetic value closest to human generated traffic

a search bar, or hyper-link. For example, the *google.com* trial begins on the *google.com* search page, where a defined text is entered. Subsequently, a user lands on the "results" page, where they are then instructed to navigate to the "images" tab, and then to click on a random image.

The five websites chosen for evaluation were *google.com*, *bbc.com*, *youtube.com*, *wikipedia.org*, and *amazon.com*. The websites were chosen for their Alexa Ranking. Further, each website represents a different category of web content, including media streaming, consumer shopping, and news. The number of websites is initially kept small due to the evaluation requirement that human generated data collection be used as a baseline. All trials were run in the `Chromium` browser, where RLBROWSE, `Selenium`, and human browsing were used to browse each website. For `Selenium`, the `Chromium` browser was run as-is, and in headless mode. `PhantomJs` was not considered, as development has ceased. Further, each browsing session shared the same browser configuration settings, with cookies disabled. For each method, 50 packet traces were taken for each website, granting a total of 1 000 packet traces. Packet traces were collected with `tshark`. Each trace lasted for 40 s, with packet collection beginning 2 s prior to loading the index page for a given trial. `Chromium` was closed 30 s after a trace started, and a 25 s delay given prior to starting the subsequent trace. All traces were collected on the same day. Human collected traces were split equally among ten participants, where participants were not informed of the 30 s window in which the browser remained open- no trial required more than the allotted time.

B. Feature Selection

Frequently used traffic features are packet size, packet timing, packet sequence, and packet direction [27]. The combination and derivation of these feature categories yields the full corpus of features- for example, combining packet size and packet direction yields the number of bytes being sent or received by a web user. A full set of features evaluated in this paper is identified in Table I.

C. Evaluation

This section focuses on the evaluation of differences in network packet trace features by automation method. Further, RLBROWSE is evaluated for improvement against observed feature disparities between `Selenium` and human generated browsing. Features are split into two categories: packet-level features, and trace-level features, where packet-level features are evaluated by their distributions over a given trace, and trace-level features are evaluated by their aggregate statistics.

We derive a measure for the difference between the two distributions p_x and q_x:

$$D_{measure}(p||q) = \frac{E[(|p(x) - q(x)|]}{2} \quad (1)$$

Equivalent distributions possess a $D_{measure}$ of 0, while a $D_{measure}$ of 1 indicates entirely dissimilar distributions. For each of the five trials, the $D_{measure}$ between the distribution of features in human generated data collection and automated data collection can be found in Table I. As the sets of non-zero bins in the distributions $p(x)$ and $q(x)$ are not equivalent, we omit the Kullback-Leibler divergence, which requires a strictly positive $q(x)$.

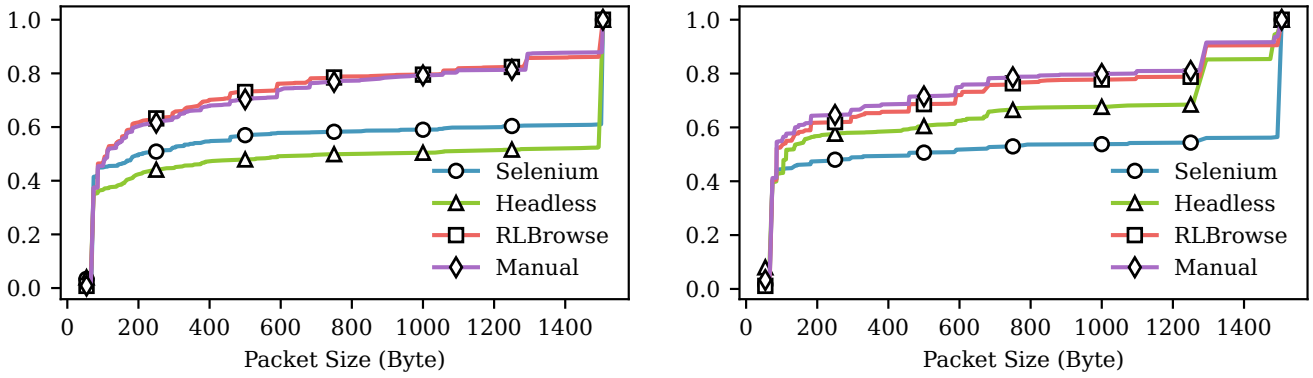
Figure 2a displays the distribution of packet sizes across all traces for each of the four data collection methods for the *wikipedia.org* trial. As expected, the majority of packets are either sent at or near MTU, such as packets carrying image resources, or contain small payloads, such as `HTML GET` requests. While packets sent with small payloads are similar in distribution across all four methods, packets sent near MTU display differing distributions. As shown, the `Selenium` and `Selenium Headless` data collection display a strongly bi-modal packet size distribution for the *wikipedia.org* trial. This is in contrast to RLBROWSE and human generated data collection, where large packets occur with a lower probability. Figure 2b displays the cumulative distribution function (CDF) of the packet sizes for the *google.com* trial, demonstrating the higher probability of larger packet sizes for the `Selenium` and `Selenium Headless` data collection. These differences are reflected in the $d_{measure}$, as shown in Table I. Comparing the $d_{measure}$ for `Selenium` and RLBROWSE to human generated data collection, we see an average of 0.242 for `Selenium`, and 0.063 for RLBROWSE, indicating significant improvement.

Feature	`Selenium`	He
Avg. Packet Size	$\alpha = 0.01$	$\alpha =$
Avg. Burst Packets Outgoing	$\alpha = 0.01$	$\alpha =$
Avg. Burst Bytes Outgoing	$\alpha = 0.01$	$\alpha =$
No. of Packets Inbound	$\alpha = 0.01$	$\alpha =$
No. of Packets Outbound	N/A*	$\alpha =$

*N/A: Weakest α tested: $\alpha = 0.10$

TABLE II: Two Variance Test for Performant RLBROWSE Features

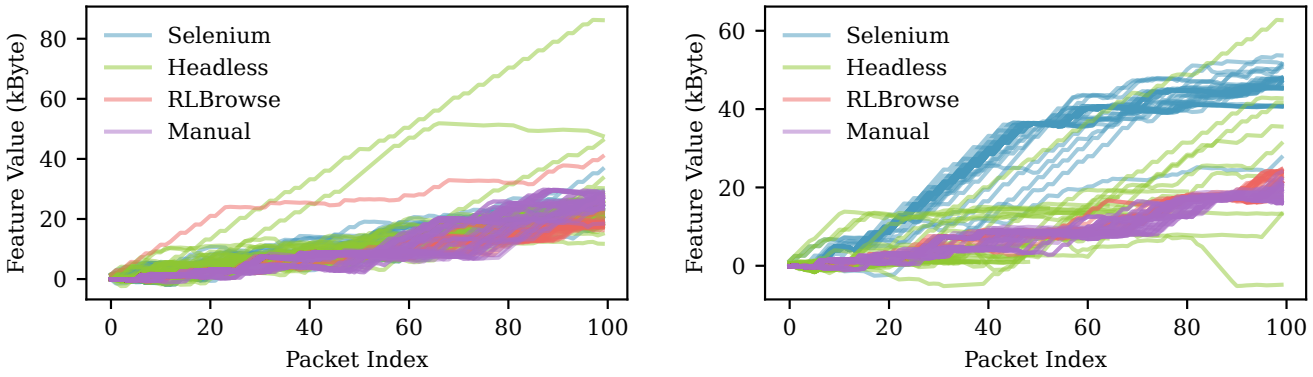
Packet size, direction, and sequence are three prevalent characteristics of network packet traces that help identify websites. Panchenko et al. [28] utilized these features to visualize a



(a) Packet Size Distribution for the *wikipedia.org* trial

(b) CDF Packet Sizes for the *google.com* trial

Fig. 2: Packet Size Distribution Disparities



(a) *bbc.com*

(b) *google.com*

Fig. 3: Packet Direction and Packet Size Fingerprints

"fingerprint" of a website- by encoding packet sizes as positive or negative, based on their direction, and then graphing the cumulative sum [28]. Based on the predictive capability of the fingerprint in classifying websites from packet traces, we visualized the differences in packet traces for `Selenium`, `Selenium Headless`, RLBROWSE, and human gathered data, as shown in Figure 3. Figure 3a shows the fingerprint for the *bbc.com* trial. Here, `Selenium Headless` shows several outliers, while the remaining methods remain grouped. However, as shown in Figure 3b, `Selenium` and `Selenium Headless` fingerprints diverge from RLBROWSE and human generated fingerprints after an average of 17 packets. The same occurs for *amazon.com* after 4 packets, and for *wikipedia.org* after 22 packets. Further, the fingerprint variances for `Selenium` and `Selenium Headless` are 191 and 203 $kByte^2$, as compared to 5 and 26 $kByte^2$ for RLBROWSE and human generated data collection.

Figure 3a demonstrates a case where the four data collection methods yielded similar results. As seen, the *bbc.com* trial shows a similar fingerprint across nearly all traces, with the exception of one outlier trace in for RLBROWSE, and three outlier traces for `Selenium Headless`. This indicates that websites are not equally affected from differences between human and synthetic trace tools. RLBROWSE improves on most of the features tested for (Table I). The average packet size differs by just 2% for RLBROWSE, while `Selenium` differs by 26%. Further, the same is true for the variance in packet sizes, at 2% and 33% respectively. Larger improvements can be seen in average outgoing packet bursts, both in the number of packets per burst, and the number of packets sent during a burst. Finally, the number of incoming and outgoing packets is also largely congruent. In Table II we analyze the variance in the synthetic datasets in relation to the variance in the human generated dataset. The chosen features are verified as normal distributions via a Chi-Square goodness of fit test. We perform a Two Variance hypothesis test for statistical features in which RLBROWSE had shown improvement. Features were calculated for each trace, yielding the tested distributions. Tests were performed at the $\alpha = 0.01$ significance level, moving to $\alpha = 0.025$ if the stronger significance level could not be met. Over the five statistical features tested, RLBROWSE possessed feature distributions such that a significance level of $\alpha = 0.10$

could not be met, indicating a higher degree of similarity with human generated traces than `Selenium` or `Selenium Headless`.

V. Conclusion

We present the RLBrowse framework, and demonstrate that disparities in network packet traces between synthetic and human generated data collection can be reduced. Further, we provide evidence that traffic trace data generated by existing web automation tools do not transfer directly to realistic web browsing.

A future iteration of the RLBrowse framework should enable the agent to find navigational destinations on its own, through the use of visual input. Further, the framework could benefit from a more versatile approach to browsing objectives, by being enabled to self-determine optimal navigation steps to reach the objective. Finally, future work should examine the benefits of RLBrowse generated data for traffic classification models.

Acknowledgements

This work is partially funded by the Federal Ministry of Education and Research in Germany (BMBF) as part of the project AI-NET-PROTECT (grant ID 16KIS1294), and the project SELMA from the "Software Campus" initiative (grant ID 01IS17049), and by the German Research Foundation (DFG) as part of the project ADVISE (grant ID - 438892507).

References

[1] Richard Barnes, Bruce Schneier, Cullen Jennings, Ted Hardie, Brian Trammel, Christian Huitema, and Daniel Borkman. Confidentiality in the face of pervasive surveillance: A threat model and problem statement. 2015.

[2] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 263–274, 2014.

[3] Inferring mechanics of web censorship around the world. In *2nd USENIX Workshop on Free and Open Communications on the Internet (FOCI 12)*, Bellevue, WA, August 2012. USENIX Association.

[4] Tarun Kumar Yadav, Akshat Sinha, Devashish Gosain, Piyush Kumar Sharma, and Sambuddho Chakravarty. Where the light gets in: Analyzing web censorship mechanisms in india. In *Proceedings of the Internet Measurement Conference 2018*, pages 252–264, 2018.

[5] Nathan Tusing, Jonathan Oakley, Geddings Barrineau, Lu Yu, Kuang-Ching Wang, and Richard R Brooks. Traffic analysis resistant network (tarn) anonymity analysis. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–2. IEEE, 2019.

[6] Jun Zhang, Yang Xiang, Yu Wang, Wanlei Zhou, Yong Xiang, and Yong Guan. Network traffic classification using correlation information. *IEEE Transactions on Parallel and Distributed systems*, 24(1):104–117, 2012.

[7] Baris Yamansavascilar, M Amac Guvensan, A Gokhan Yavuz, and M Elif Karsligil. Application identification via network traffic classification. In *2017 International Conference on Computing, Networking and Communications (ICNC)*, pages 843–848. IEEE, 2017.

[8] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1928–1943, 2018.

[9] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. *arXiv preprint arXiv:1708.06376*, 2017.

[10] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 1187–1203, 2016.

[11] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, pages 407–414. sn, 2016.

[12] Zhong Fan and Ran Liu. Investigation of machine learning based network traffic classification. In *2017 International Symposium on Wireless Communication Systems (ISWCS)*, pages 1–6. IEEE, 2017.

[13] Mohammed Elnawawy, Assim Sagahyroon, and Tamer Shanableh. Fpga-based network traffic classification using machine learning. *IEEE Access*, 8:175637–175650, 2020.

[14] Zhiyong Bu, Bin Zhou, Pengyu Cheng, Kecheng Zhang, and Zhen-Hua Ling. Encrypted network traffic classification using deep and parallel network-in-network models. *IEEE Access*, 8:132950–132959, 2020.

[15] Ariya Hidayat et al. Phantomjs. *Computer software. PhantomJS. Vers*, 1(7), 2013.

[16] Antoine Vastel, Walter Rudametkin, Romain Rouvoy, and Xavier Blanc. Fp-crawlers: studying the resilience of browser fingerprinting to block crawlers. In *MADWeb'20-NDSS Workshop on Measurements, Attacks, and Defenses for the Web*, 2020.

[17] Shaon Barman, Sarah Chasins, Rastislav Bodik, and Sumit Gulwani. Ringer: web automation by demonstration. In *Proceedings of the 2016 ACM SIGPLAN international conference on object-oriented programming, systems, languages, and applications*, pages 748–764, 2016.

[18] Tessa Lau, Julian Cerruti, Guillermo Manzato, Mateo Bengualid, Jeffrey P Bigham, and Jeffrey Nichols. A conversational interface to web automation. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, pages 229–238, 2010.

[19] Greg Little, Tessa A Lau, Allen Cypher, James Lin, Eben M Haber, and Eser Kandogan. Koala: capture, share, automate, personalize business processes on the web. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 943–946, 2007.

[20] Hanlin Chen, Hongmei He, and Andrew Starr. An overview of web robots detection techniques. In *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–6. IEEE, 2020.

[21] Qilei Yin, Zhuotao Liu, Qi Li, Tao Wang, Qian Wang, Chao Shen, and Yixiao Xu. Automated multi-tab website fingerprinting attack. *IEEE Transactions on Dependable and Secure Computing*, 2021.

[22] Luca Vassio, Idilio Drago, Marco Mellia, Zied Ben Houidi, and Mohamed Lamine Lamali. You, the web, and your device: Longitudinal characterization of browsing habits. *ACM Transactions on the Web (TWEB)*, 12(4):1–30, 2018.

[23] Tomasz Wesolowski, Malgorzata Palys, and Przemyslaw Kudlacik. Computer user verification based on mouse activity analysis. In *New Trends in Intelligent Information and Database Systems*, pages 61–70. Springer, 2015.

[24] Maja Pusara and Carla E Brodley. User re-authentication via mouse movements. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 1–8, 2004.

[25] Iraklis Kordomatis, Christoph Herzog, Ruslan R Fayzrakhmanov, Bernhard Krüpl-Sypien, Wolfgang Holzinger, and Robert Baumgartner. Web object identification for web automation and meta-search. In *Proceedings of the 3rd International Conference on Web Intelligence, Mining and Semantics*, pages 1–12, 2013.

[26] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[27] Junhua Yan and Jasleen Kaur. Feature selection for website fingerprinting. *Proc. Priv. Enhancing Technol.*, 2018(4):200–219, 2018.

[28] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website fingerprinting at internet scale. In *NDSS*, 2016.