2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and
Networks (CAMAD)

Catching free-riders: in-network adblock detection with machine learning techniques

Daniele Moro[*†], Filippo Benati[*†], Michele Mangili[†], Antonio Capone[*]
[*]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy
[†]ErnieApp Ltd, Dublin, Ireland
daniele.moro@polimi.it - filippo.benati@mail.polimi.it - michele@ernieapp.com - antonio.capone@polimi.it

***Abstract*—The rise of adblockers is creating lots of concerns to the online content publishing industry, as it severely affects the possibility to offer free-content to end-users by subsidizing the fruition costs with advertisements.**

While many detection techniques have been proposed as a countermeasure to the diffusion of adblocks, they either rely on the injection of code in the served web pages, or require to perform passive measurements for a long time, thus leading to high costs and delays before collecting the desired information.

Motivated by these reasons, in this paper we propose a novel technique to conduct in-network adblock usage measurements, inspecting only few minutes of network traffic. Our approach relies on network traffic inspection, and classification with machine learning techniques to detect whether the user is blocking, or not, the advertisements.

Key findings obtained show that by inspecting only few minutes of network traffic, we can reliably perform the detection with an accuracy up to 99%, with a negligible computational overhead.

I. Introduction

Recent reports estimate that $12 billion dollars in ad revenues will be lost in 2020 accountable to the usage of adblockers [1]. This problem is severely affecting the ad-supported free online publishing industry [2], which attempted to respond by preventing end-users to load web-pages whenever they appear to be blocking advertisements [3].

Nowadays, adblock detection is performed with *anti-adblock* tools embedded in web-pages as JavaScript code, and the same technology is also adopted to compute adblock usage statistics for a given website [4]. However, the adblockers community reacted by crafting new filtering lists that can now block anti-adblock scripts themselves, thus giving birth to *anti-"anti-adblock"* tools [5]. Thanks to them, not only end users can by-pass anti-adblocks and still access the desired content, but they also impact the quality of the adblock usage statistics leading to false negatives and making it complex to reliably know adblock diffusion.

Furthermore, reusing the same script-based technology to extract adblock usage reports on a country or worldwide scale is even more challenging. As a matter of fact, it would be necessary to cover a large sample of popular websites in the considered geographic region, partnering with publishers to include the detection scripts in all the served web-pages, and then obtain usage data. For all these reasons, recent popular reports have computed adblock adoption statistics either relying on indirect metrics or by leveraging user surveys [6].

Despite that adblock, anti-adblock and anti-"anti-adblock" are tools characterized by an increasing level of complexity, for the sake of measuring adblock usage, they all share common denominators (*and weak-spots*): first of all they usually work within the scope of end-users' browsers, making it difficult to extract general statistics independently from the website visited, and secondly, they normally rely on the usage of filter lists which need to be kept constantly (and mutually) updated, thus having high maintenance costs [7], [8].

To overcome these challenges, previous scientific works performed in-network adblock usage analysis by inspecting network traffic and detecting evidences that the user is using (or not) an adblock [9], [10]. These studies have clearly shown that the usage of adblocks has an impact on the network traffic. However, to perform the adblock detection, they either inspected network traffic in clear text [9], making it difficult to apply similar techniques to encrypted traffic, or checked for adblock update requests [10], introducing a non-negligible delay to inspect long-lasting traces for each customer and wait for the adblock update process to happen.

Motivated by all these reasons, in this paper we propose a new approach to measure adblock usage on a large scale, passively inspecting only few minutes of network traffic and then leveraging machine learning to classify the flows. Our approach settles on a simple intuition: rather than testing for content filtering within the scope of a web-page rendered in a browser, we observe and compare the rate of network requests originating from advertising services, with those that do not carry ads. While previous works [9], [10] have observed that this rate changes as a result of using an adblock, it is challenging to set a threshold on the number of requests in an empirical way, and this is the reason why our approach leverages machine learning techniques to perform the training and classification in an automatic fashion. Our tool supports large-scale analysis, and generates fine-granular user-level data, that can be helpful not only to generate adblock usage reports, but also to provide useful insights when choosing how to allocate advertising investments.

We designed our scheme from the ground up to overcome the major shortcomings affecting current state-of-the-art techniques for adblock detection, in particular our proposal is *website-agnostic*, as it works across all websites without requiring to have a partnership with all the owners to inject JavaScript code in the pages; it is *platform-independent*, as it

978-1-5386-6151-2/18/$31.00 ©2018 IEEE

can seamlessly detect adblocks for desktop or mobile devices; it is *efficient*, as it needs to inspect only few minutes of traffic, and finally it is *update-tolerant*, as it can react to any update to adblocks, anti-adblocks, and anti-"anti-adblock" filtering lists, requiring no human intervention at all.

To summarize, in this paper we provide the following contributions:

1) we propose a novel in-network adblock detection technique to conduct large scale adblock-usage analysis in a cost effective manner;
2) we extend our base classification scheme by adding (a) a *pre-classifier* to filter-out idle-traffic records, and (b) a *post-classifier* to perform the adblock detection only at specific timeslots with high classification probability;
3) finally we perform extensive numerical evaluation of the classifiers under several conditions, for both desktop and mobile traffic.

Our key findings suggest that not only the adblock classifier itself can reach a high classification accuracy up to 95%, performing the detection for millions of users in just few seconds, but we can also further boost the classification accuracy up to 99%, by *pre-* and *post-*filtering the observations.

The remainder of this paper is organized as follows. Sec. II discusses related works. Sec. III presents the methodology used, describing the challenges addressed by our proposal. Numerical results and key findings are illustrated in Sec. IV, and finally concluding remarks are presented in Sec. V.

II. Related Work

Current state-of-the-art adblock detection tools leverage JavaScript code to test whether the end user's browser interferes with normal content loading [3], [4], [7], [8]. More in detail, detection scripts check whether bait content that would normally be blocked by adblock "black-lists" (for example, a file named *ads.js*), can be loaded in the current page [4]. If this fake content cannot be retrieved, the user is indeed blocking advertisement, and specific reactions can be performed, from the mere reporting for analytics, up to denying access to the web-page. A similar approach has been used in [2], where authors estimate the fraction of users using adblock by comparing with a Mixture Proportion Estimation technique the data collected as a ground truth from a panel of users.

By inspecting network traffic, the main advantage of our approach is that it does not require to embed code in any web page, and therefore can be used to implement transparently the adblock detection functionality for a large set of websites and users.

Similarly to our approach, in [9] Pujol et al. leveraged the *Bro* HTTP analyzer to extract HTTP information in clear text and then classify flows using *libadblockplus* (a library that implements the core functions of Adblock Plus [11]) distinguishing between ads and normal traffic requests. Metwalley et al. in [10] performed an analysis of online tracking by leveraging passive measurement, and studying the number of third-party services contacted when surfing the web. Both [9] and [10] implemented adblock detection by extracting adblock updates from the collected trace.

Given that adblock updates are usually triggered at sporadic time intervals (usually once every 4 or even more days), these approaches introduce the need of capturing long traffic traces to look for the adblock update flows. On the other hand, our approach does not suffer from this shortcoming, as it can detect adblock usage just by inspecting few minutes of network traffic.

Several studies have analyzed anti-adblock scripts, providing tools to seamlessly detect their execution in HTML pages. In [12], Mughees et al. analyzed the most common reactions of adblock detectors, and trained a machine learning classifier to identify anti-adblock code. Nithyanand et al. surveyed in [7] popular websites employing adblock detection techniques, while in [3] the authors presented a differential execution analysis approach to discover anti-adblockers.

Anti-adblock detection made possible to create anti-adblock filter lists (e.g., [5], [13]), which can be used by adblocks to transparently bypass the adblock-detection scripts themselves. In [8] the authors present a preliminary analysis on anti-adblock filter lists, highlighting that their update process suffers from major limitations, namely the fact that any change to an adblock, anti-adblock or anti-adblock filter may potentially trigger a ripple effect, forcing each provider to keep his tools constantly updated with respect to all the adversaries.

Contrary to previous proposals, our approach is specifically tailored to conduct large scale adblock usage analysis in a cost-effective manner, overcoming the common weak spots listed above. First of all, our approach is *list-less*, and does not require manual intervention to keep the algorithm updated, secondly it is unaffected by the usage of *anti-adblock* lists, since it only analyzes the network flows that carry advertising contents, and finally it does not require to embed any code in a web-page, and therefore it is *website-agnostic*.

III. Methodology and Approach

This section illustrates the software components we designed and used to perform the analysis and to generate data. The full architecture is depicted in Fig. 1 and can be functionally split in three major components, namely *Data Collection* (discussed in Sec. III-A) which generates synthetic traces and labels them, *Features Extraction* (Sec. III-B) used to perform traffic inspection producing the feature dataset, and *Machine Learning Classification* (Sec. III-C) which, using the features generated at the previous step, implements the chain of classifiers.

A. Data Collection

In order to collect data to train the classifiers, we opted for generating traffic traces automatically, implementing a web-crawler with *Selenium*[1] and *Appium*[2], as similarly done in other notable research works [9], [12]. This solution has many advantages compared to using real users' traffic. In fact,

[1] https://www.seleniumhq.org/

[2] A framework for mobile apps automation. Website: http://appium.io/

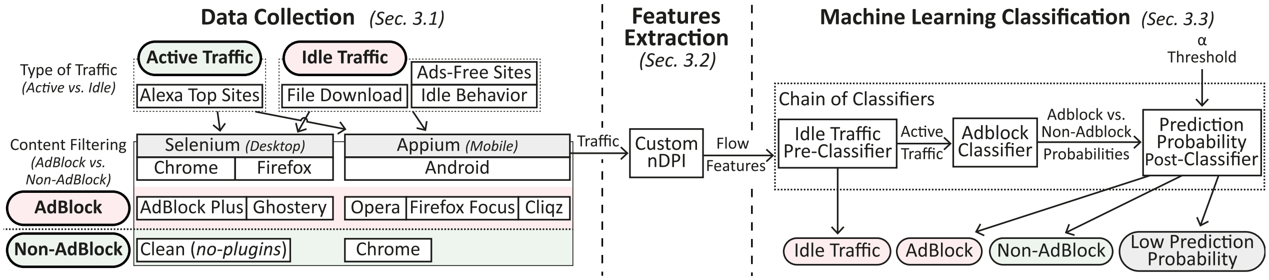


Figure 1: Proposed architecture for data collection with synthetic traffic generation, flow-level features extraction and machine-learning classification.

by spawning many crawlers on virtual machines running in data-centers in different geographic regions, not only a large amount of data can be generated in few hours at a very low cost, but we can also better map ads providers on a global scale. Furthermore, we can easily generate synthetic traces with different browsers and filtering plugins, just by changing the crawler configuration, as shown in Fig. 1.

We instructed the crawler to visit the homepage of the most popular websites worldwide, with a frequency distribution equal to their popularity. In particular, we crawled the first 1000 websites appearing in the Alexa Top list[3], a representative data source of websites popularity used also in other works [7], [14].

To better mimic the standard user's behavior, we refined the crawler for Google Search, Twitter, YouTube, Facebook, Instagram and Twitch, by making it perform specific actions, such as playing videos published on the most popular YouTube channels, or performing queries on Google search, as well as opening influencers' profiles on social networks.

For the desktop scenario we collected data with *Chrome* and *Firefox*, because of their market share [15], and we also took into consideration the effects of different filtering policies by using Adblock Plus [11] and Ghostery [16], two of the most popular adblock and anti-tracking tools available for both browsers [8]. As depicted in Fig. 1, no content-filtering was used with the *Clean* plugin profile, while *AdBlock Plus* and *Ghostery* were both blocking advertising. For the mobile scenario, we collected data using Opera for Android (configured to block ads), Firefox Focus, Cliqz and the Google Chrome Mobile (used as a reference for non-blocking ads).

Lastly, we complemented our Data Collection component by instructing the crawler to generate traffic traces to simulate *"idle"* users' behavior, as further discussed in Sec. III-C, IV-B and IV-C. For this case, we instructed the crawler to either visit websites that do not provide advertisements, or to download files images and documents, as well as to simply generate background traffic produced by the operating system and application updates.

B. Feature Extraction

Features extraction was performed by analyzing network traffic with a customized version of nDPI [17], an open-source Deep Packet Inspection probe.

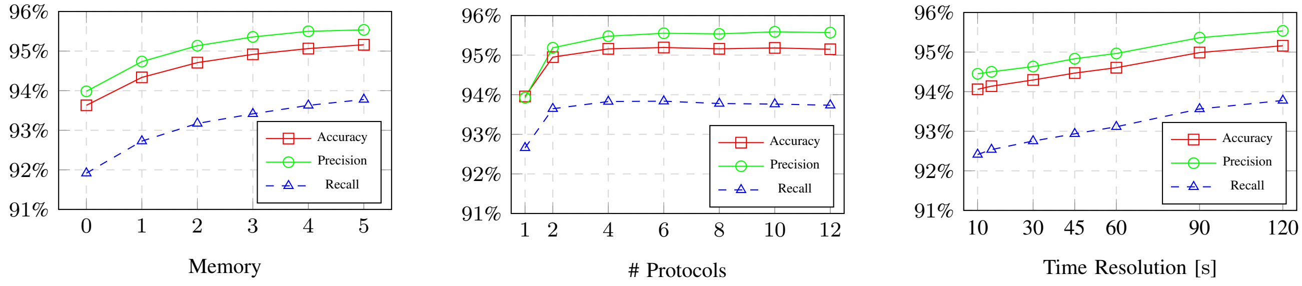
A fundamental information that we wanted to extract from the DPI is whether a flow carries (or not) *"advertisement"* content. Although nDPI can nowadays detect hundreds of protocols, the current latest version (v2.2) cannot yet discern whether a flow is an ad, or not. To overcome this limitation, we extended nDPI, testing whether the detected hostname, or the Server-Name Indication (SNI) matched one of those included in popular adblock filter lists (e.g., [18], [11]). Whenever there is a match, we tag the network flow with the *"ADS"* label, otherwise we tag it as being *"NON_ADS"*.

Our approach is not influenced by encrypted traffic since we are not analyzing the packets payload but only the HTTP/HTTPS headers that are un-encrypted. Recent proposal [19] describes the possibility to encrypt also the SNI field in HTTPS headers (which is a fundamental information that we used). However, the new TLS 1.3 standard does not include SNI encryption (even if it includes the encryption of the connection handshake), thus our approach is still viable with the new standards [20]. Moreover, SNI is widely adopted for HTTPS traffic as it removes the need of having a dedicated IP address for each certificate, in fact, as shown in recent reports (e.g.: [21]), SNI adoption is well above 99%.

In addition to our custom advertising tag, we also extracted the application/service to which the flow belongs, as detected by nDPI (called *protocol*). To keep the number of features contained, rather than considering all the hundreds protocols/applications supported by nDPI, we restricted our analysis to a subset of 11 of them, including the most popular services (i.e. Google / Facebook / Twitter / YouTube / Amazon / eBay / Instagram / Microsoft / Yahoo / Cloudflare). On top of these, we added the *"OTHERS"* meta-protocol, which groups all the network flows not belonging to the applications considered in the analysis.

For each network flow, we extracted from nDPI the following flow-level metrics: 1) The detected protocol; 2) The ADS / NON_ADS tag; 3) The number of packets; 4) The volume of traffic (in bytes). Even though we implemented our system customizing nDPI, our approach is DPI-agnostic and can therefore be applied to other Deep Packet Inspection tools as well, including commercial products. Our approach can be applied also by network operators which usually have already deployed DPI probes in their Point-of-Presence (POP).

[3] https://www.alexa.com/topsites



(a) Effect of Memory

(b) Effect of the Number of Protocols

(c) Effect of Resolution

Figure 2: Accuracy, precision and recall of the *adblock classifier*.

We generate the features splitting the time into slots at a fixed resolution, and then aggregating and normalizing the statistics. We also evaluated the impact of memory, by adding to each features vector the value of the statistics observed in previous time-slots. We considered a varying number of memory steps from 0 (*no-memory*) up to 5 previous slots.

C. Machine Learning Classification

While having an impact on flows, as observed in other works such as [9], [10], performing adblock detection inspecting only few minutes of network traffic is far from being an easy task. For this reason, we decided to apply a Machine Learning classifier to the extracted features vector as to be able to automatically compute thresholds on the extracted metrics, to identify whenever the user is using the ad-block or not by jointly considering a large set of network-level metrics.

While running preliminary analysis we discovered that there are specific cases in which the information arising from the inspection of the flows is not sufficient to perform adblock detection. Notable use-cases are: 1) when the user is not actively browsing, but the device generates *background traffic*; 2) when the user is browsing on websites that are *ads-free*[4], or 3) when the user is simply *downloading* files (images, documents etc.).

Letting the classifier perform the classification even in these adverse cases leads to poor classification performance. For this reason, the Machine Learning Classification component we designed is characterized by a chain of three major steps (as in Fig. 1): (1) a *pre-classifier*, (2) the *adblock classifier*, and (3) a *post-classifier*.

More in detail, the objective of the *pre*-classifier is to discriminate between *active* and *idle* traffic (intrinsically ads-free traffic), filtering records belonging to the latter class out of the subsequent steps of the classification chain.

Once that only "active" traffic is preserved, the data record is then analyzed by the adblock classifier, which normally outputs probability values for the AdBlock and Non-AdBlock class.

Finally, the *post*-classifier filters the output of the adblock classifier, based on the obtained probability values and the α-threshold parameter. When the probability value of either the "clean" or the "adblock" class generated by the adblock classifier is below α, the record is discarded for having *"Low Prediction Probability"*.

By using this chain of classifiers we were able to boost the classification performance up to 99% accuracy, as thoroughly discussed in Sec. IV-B and Sec. IV-C.

IV. Evaluation and Discussion

In this section we analyze and discuss the numerical results obtained performing extensive analysis under realistic traffic conditions.

All the tests were performed using Python 3.5.1 and Scikit learn v0.19.1 on AWS EC2 r3.xlarge instances[5], having 4 vCPUs, 30.5 GB RAM and 1x80GB SSD. nDPI inspected 1TB of traffic generated with our proposed Data Collection system. The traffic was generated from different AWS regions (Asia, USA, South America and Europe) to emulate traffic coming from all over the world. Unless stated otherwise, the numerical results refer to the following base configuration: *Random Forest classifier* (RF), *5 memory slots*, *8 protocols*, *120 seconds* slotting resolution.

Training and testing was performed using multiple random *Shuffle & Splits* with 70% of the dataset used for training and 30% for testing. The performance of the classifier is then evaluated averaging the results from 20 runs and according to accuracy, precision and recall metrics. On the graphs we show the precision and recall as being the averages of the "clean" and "adblock" classes.

A. Adblock Classifier Performance

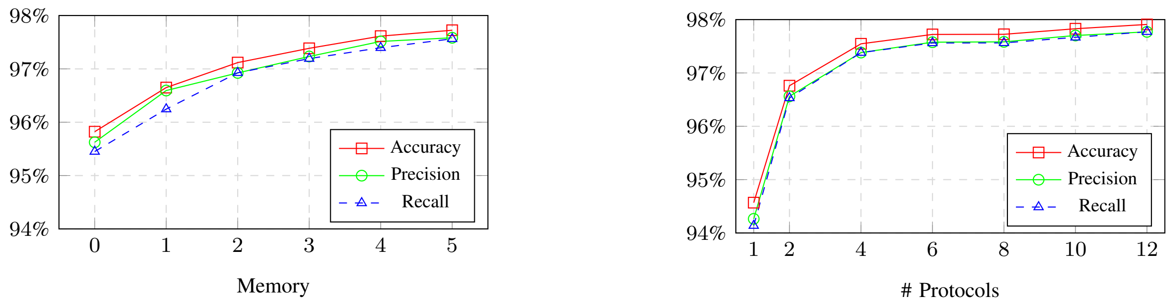
This section discusses the results depicted in Fig. 2 (which refers to the "adblock classifier" only), generated with both *active* and *idle* traffic.

After testing commonly used classifiers (Decision Trees, Naïve Bayes, Random Forest and Neural Networks classifiers), as expected, we obtained the best results with a Random Forest classifier (with 10 trees without pruning), reaching 95% accuracy and precision, and 92% recall.

The effect of memory is illustrated in Fig. 2a, where we observe that the higher the memory, the better the results.

[4] Wikipedia is one notable example, see https://donate.wikimedia.org/

[5] https://aws.amazon.com/it/ec2/instance-types/



(a) Effect of Memory

(b) Effect of Protocols

Figure 3: Accuracy, precision and recall of the *pre-classifier*.

More in detail, we obtained an improvement of at least 2% for all performance metrics, just by adding 5 memory slots, compared to the case using only current data (i.e. the *zero-memory* scenario). Indeed, the classifier can take advantage of the observations in previous time slots, checking whether the user is systematically blocking advertising or whether it was just a sporadic event.

Being able to discriminate a large number of application level protocols with the DPI has only a minor impact on the adblock classification results, as shown in Fig. 2b. More in detail, when the number of protocols is equal to 1, the DPI detects all flows as belonging to the *"OTHER"* protocol class, described in Sec. III-B, and therefore the classification only relies on the ADS / NON_ADS tag. Rather than being able to classify many different protocols, for our use-case, it is better to focus only on few of them, usually those that drive most of the traffic (e.g., *Google / Facebook / YouTube*).

Fig. 2c shows the effect of the slotting resolution. We observed that inspecting only 10 seconds of traffic is sufficient to reach an accuracy higher than 94%, while it is possible to reach 95% accuracy by observing 120 seconds of traffic.

Lastly, we tested the average time needed to train the classifier and the time needed to classify 1 million data records. Training a Random Forest classifier with all the collected dataset required less than 4 seconds, while we were able to do the classification of 1 million of records in less than 8 seconds, showing that our proposed scheme can scale up to large-case scenarios.

B. Pre-classifier Performance

This section presents the results observed for the *pre-classifier*, shown in Fig. 3.

As described in Sec. III-C the objective of the pre-classifier is to filter out *"idle"* traffic (i.e.: traffic intrinsically ads-free, such as background traffic, ads-free websites, file and document download), which cannot be used as such to discern whether the user is using an adblock or not. On the other hand, non-idle traffic likely contains relevant data for the purpose of adblock detection, and therefore it can be used in subsequent steps of the classification chain.

For the pre-classifier we still used a Random Forest (with 10 trees and no pruning), as in Sec. IV-A, but this time we trained it to discriminate idle and non-idle traffic. Overall, we observed a high pre-classification accuracy (up to 97.7%), as well as high precision and recall, meaning that there is no bias between the classification performance of the *active* and *idle* traffic class, as in Fig. 3.

The effect of memory is shown in Fig. 3a, where we observe that increasing memory up to 5 slots we obtained a 2% increment on all the performance metrics, as similarly observed in Sec. IV-A. On the other hand, Fig. 3b shows the impact of the number of protocols identified by the DPI. Similarly to what observed in Sec. IV-A, by introducing in the features vector metrics for other application layer protocols an improvement of 3.5% in classification accuracy is observed. This is accountable to the fact that by introducing more protocols, the classification algorithm can better discriminate whether the user is generating idle or active traffic.

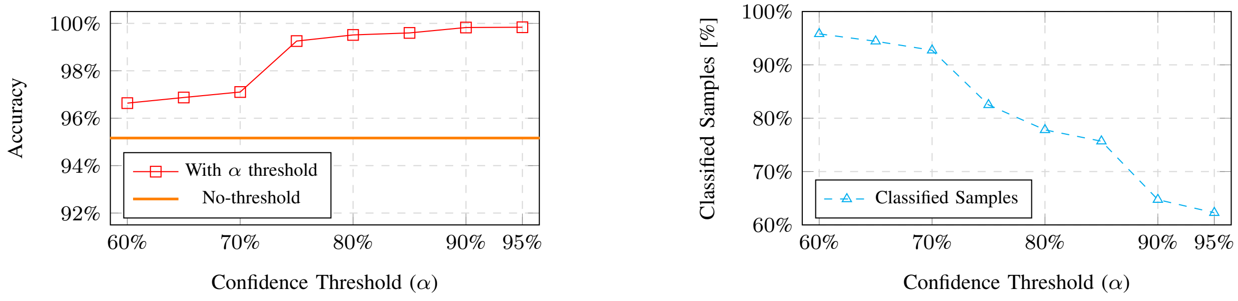
C. Post-classifier Performance

This section presents the results observed for the *post-classifier* which are depicted in Fig. 4. In particular, Fig. 4a compares the accuracy result obtained with the bare adblock classifier, with the post-classifier data, for different α values. As expected, we observe that the higher the α-value, the higher the accuracy, since the classifier only performs the decision with very high classification probabilities. However, this comes as a trade-off with the number of samples discarded by the classifier, as shown in Fig. 4b. For the considered scenarios we observed a good compromise setting $\alpha = 75\%$, for which we obtained 99.2% classification accuracy, while discarding less than 20% of the samples.

D. Mobile Traffic Performance

In this section, we focus our analysis on the mobile scenario, and present the testing results obtained with mobile-traces generated with Appium and containing, for the sake of simplicity, "active" traffic traces only.

Tab. I summarizes the values observed (with the base settings reported at the beginning of Sec. IV). As expected, given that we fed the chain with *"active"* records-only, less than 200 samples were filtered by the pre-classifier, leading to 99% accuracy of this initial step. Although the adblock-detector scored 84.4% accuracy, by setting $\alpha = 75\%$ and



(a) Effect on accuracy

(b) Effect on samples

Figure 4: Effect of α on post classification.

Training set	1042634 samples
Testing set (mobile only)	21278 samples
Pre-classifier filtered samples	198 (99% accuracy)
Adblock detector accuracy	84.4%
Confidence threshold α	75%
Out of threshold samples	11498 (54% dropped)
Detection accuracy after threshold	96.2%

Table I: Mobile traffic results

then running the post-classification, we could improve the accuracy up to 96.2%, dropping half of the samples. In the considered scenario *post*-classification dramatically improves the accuracy of almost 12%, at the minor cost of doubling the observation window. Rather than taking immediately the classification decision, it is better to wait (almost 5 minutes in the considered example), and perform the detection only at specific time slots where the quality of the data is higher.

V. Conclusion

In this paper we tackled the problem of detecting adblock usage from the network stand point. In order to do that we exploited traffic inspection and machine learning techniques to detect when the user is blocking, or not, advertisements.

Our in-network approach goes beyond current state-of-the-art adblock detection techniques making it possible to conduct large-scale adblock usage measurements by inspecting only few minutes of network traffic for each user. In particular, numerical results have shown that our tool can reach very high accuracy up to 95%, which can further be increased up to 99% by adding a pre- and post-classification steps, to filter time slots for which the inspected traffic did not carry enough information to perform the decision.

References

[1] "Optimal.com/Wells Fargo Securities Ad Blocking Survey," http://optimal.com/optimal-com-wells-fargo-ad-blocking-survey/, 2016, last accessed: May 2018.

[2] M. Malloy, M. McNamara, A. Cahn, and P. Barford, "Ad blockers: Global prevalence and impact," in *Proceedings of the 2016 Internet Measurement Conference*. ACM, 2016, pp. 119–125.

[3] S. Zhu, X. Hu, Z. Qian, Z. Shafiq, and H. Yin, "Measuring and disrupting anti-adblockers using differential execution analysis," in *Proc. of the Network & Distributed System Security Symposium (NDSS), San Diego, CA, USA*, 2018.

[4] "Detect Adblock," https://www.detectadblock.com/, 2018, last accessed: May 2018.

[5] "Anti-Adblock Killer List," https://reek.github.io/anti-adblock-killer/, 2018, last accessed: May 2018.

[6] PageFair, "Adblock Report," https://tinyurl.com/ybhw5fxf/, 2017, last accessed: May 2018.

[7] R. Nithyanand, S. Khattak, M. Javed, N. Vallina-Rodriguez, M. Falahrastegar, J. E. Powles, E. Cristofaro, H. Haddadi, and S. J. Murdoch, "Adblocking and counter blocking: A slice of the arms race," in *CoRR*, vol. 16. USENIX, 2016.

[8] U. Iqbal, Z. Shafiq, and Z. Qian, "The ad wars: retrospective measurement and analysis of anti-adblock filter lists," in *Proceedings of the Internet Measurement Conference (IMC)*. ACM, 2017, pp. 171–183.

[9] E. Pujol, O. Hohlfeld, and A. Feldmann, "Annoyed users: Ads and ad-block usage in the wild," in *Proceedings of the 2015 Internet Measurement Conference*. ACM, 2015, pp. 93–106.

[10] H. Metwalley, S. Traverso, M. Mellia, S. Miskovic, and M. Baldi, "The online tracking horde: a view from passive measurements," in *International Workshop on Traffic Monitoring and Analysis*. Springer, 2015, pp. 111–125.

[11] "AdBlock Plus," https://adblockplus.org/, 2018, last accessed: May 2018.

[12] M. H. Mughees, Z. Qian, Z. Shafiq, K. Dash, and P. Hui, "A first look at ad-block detection: A new arms race on the web," *arXiv preprint arXiv:1605.05841*, 2016.

[13] "Adblock Warning Removal List," https://easylist-downloads.adblockplus.org/antiadblockfilters.txt, 2018, last accessed: May 2018.

[14] H. Jelodar, Y. Wang, C. Yuan, and X. Jiang, "A systematic framework to discover pattern for web spam classification," *arXiv preprint arXiv:1711.06955*, 2017.

[15] "Browser Worldwide Market Share from GlobalStats," http://gs.statcounter.com/, 2018, last accessed: May 2018.

[16] "Ghostery," https://www.ghostery.com/, 2018, last accessed: May 2018.

[17] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, "ndpi: Open-source high-speed deep packet inspection," in *Proc. of the IEEE Int.l Wireless Communications and Mobile Computing Conference (IWCMC)*, 2014, pp. 617–622.

[18] "Steven Black Hosts List," https://github.com/StevenBlack/hosts, 2018, last accessed: May 2018.

[19] "Issues and Requirements for SNI Encryption in TLS," https://www.ietf.org/id/draft-ietf-tls-sni-encryption-03.txt, last accessed: May 2018.

[20] Sandvine, "Encryption and DPI: Current and Future Services Impact," https://www.sandvine.com/hubfs/downloads/solutions/analytics-and-insights/sandvine-wp-encryption-and-dpi.pdf.

[21] E. Nygren, "Encrypting the Web for All: The Need to Support TLS SNI in the Remaining Clients," https://developer.akamai.com/blog/2017/10/20/encrypting-web-need-support-tls-sni-remaining-clients/, last accessed: May 2018.