Like a Pack of Wolves: Community Structure of Web Trackers

Vasiliki Kalavri[1], Jeremy Blackburn[2], Matteo Varvello[2], and Konstantina Papagiannaki[2]

[1] KTH Royal Institute of Technology
[2] Telefonica Research

Abstract. Web trackers are services that monitor user behavior on the web. The information they collect is ostensibly used for customization and targeted advertising. Due to rising privacy concerns, users have started to install browser plugins that prevent tracking of their web usage. Such plugins tend to address tracking activity by means of crowdsourced filters. While these tools have been relatively effective in protecting users from privacy violations, their crowdsourced nature requires significant human effort, and provide no fundamental understanding of how trackers operate. In this paper, we leverage the insight that fundamental requirements for trackers' success can be used as discriminating features for tracker detection. We begin by using traces from a mobile web proxy to model user browsing behavior as a graph. We then perform a transformation on the extracted graph that reveals very well-connected communities of trackers. Next, after discovering that trackers' position in the transformed graph significantly differentiates them from "normal" vertices, we design an automated tracker detection mechanism using two simple algorithms. We find that both techniques for automated tracker detection are quite accurate (over 97%) and robust (less than 2% false positives). In conjunction with previous research, our findings can be used to build robust, fully automated online privacy preservation systems.

1 Introduction

The massive growth of the web has been funded almost entirely via advertisements shown to users. Web ads have proven superior to traditional advertisements for several reasons, the most prominent being the ability to show personally relevant ads. While the content of the web page the ad is being served on can help provide hints as to relevance, web advertisement agencies also rely on mechanisms to *uniquely identify* and *track* user behavior over time. Known as *trackers*, these systems are able to uniquely identify a user via a variety of methods and over time can build up enough information about a user to serve extremely targeted ads.

While ad agencies' use of trackers has enabled services to provide access to users free of charge, there is also a certain degree of "creepiness" in the way the current ecosystem works that has also been highlighted in the US congress [12]. Recent work [5] has even shown that government agencies can easily exploit

trackers to spy on people. Privacy concerns have led to the creation of client side applications that block trackers and ads. For example, AdBlock [1] blocks trackers by filtering requests through a set of crowdsourced rules. Unfortunately, such lists are mostly opaque: there is no straight forward way to understand why a tracker was added to the list or to get a sense as to how trackers work on an individual or group basis, and users can be left out in the cold as evidenced by the recent sale of AdBlock to an undisclosed buyer who immediately enabled opting in to the "Acceptable Ads" program [14].

In the research community, several strategies for detecting and defending against trackers have been introduced [7, 10, 13]. Overall, these works focus on understanding the methods that trackers use in order to define techniques for obfuscating a user's browsing behavior. However, these previous works are generally focused on lower level intricacies, e.g., how trackers fingerprint users or ensure that cookies persist even after users clean them.

In this paper, we take a different approach and attempt to characterize some more fundamental aspects of trackers. Our rationale is that user requests, e.g., accessing `google.com`, and requests to trackers, e.g., 3rd party request to `doubleclick.net`, can be represented as a bipartite graph from which we can derive unique tracker properties, allowing for the optimization and automation of the tracker detection problem.

This work makes several contributions.

1. We model user browsing as a 2-mode graph using 6 months (November 2014 - April 2015) of traffic logs from an explicit web proxy. By analyzing this graph, we discover that trackers are very well connected: 94% appear in the largest connected component of the graph.
2. We explore the communities trackers form by inducing a 1-mode projection of the 2-mode browsing graph. We find that trackers form very well-defined communities that distinguish them from regular URLs.
3. We show that the 1-mode projection graph is a useful tool to automatically classify trackers with high precision and very low false positive rate. More importantly, using the projection graph for tracker detection is very robust to evasion since it captures a fundamental necessity of the tracking ecosystem: presence of trackers on multiple sites, and presence of multiple trackers on the same site, which allows publishers to better monetize ad display through real time bidding. Changing such a behavior would limit the efficiency of tracking as a whole.

2 Background and Dataset

Trackers enable targeted advertising and personalization services by monitoring user behavior on the web. To understand web tracking, let us consider what happens in the browser when a user visits a URL. First, the browser issues an HTTP request to the site to fetch the contents of the web page. The response contains the page resources, including HTML, and references to embedded objects like images and scripts. These references might then instruct the browser to make additional HTTP requests (e.g., for the image itself) until the page is fully loaded. Embedded objects can be hosted on different servers than the page

content itself, in which case they are referred to as *third-party* objects. A fraction of these third-party objects open connections to trackers, e.g., the popular Facebook "like" button, at which point the users' online whereabouts are logged for targeting/personalization purposes.

2.1 Dataset

Our dataset is derived from 6 months (November 2014 - April 2015) of traffic logs from an explicit web proxy. The proxy is operated by a major telecom located in a large European country. Our data is delivered to us in the form of augmented Apache logs. The logs include fields to identify the user that made the access, the URL that was requested, headers, performance information like latency and bytes delivered. We call this dataset the *proxy log*, and in total it represents 80 million accesses to 2 million individual sites. In the following section, we describe how we use the proxy log to model web tracking as a graph problem. We label URLs in our dataset as *tracker* or *other* based on ground truth derived from the EasyPrivacy list for AdBlock [3].

2.2 Web Tracking as a Graph Problem

A *2-mode graph* is a graph with two different modes (or classes) of vertices, where edges are only allowed between vertices belonging to different modes. The interactions between explicit user requests and background requests, both page content and third-party objects like web tracking services, can be naturally modeled as a 2-mode graph. The first mode of vertices in the graph are URLs that the user intentionally visits, while the second mode are URLs for objects that are embedded in the visited page.

More precisely, we represent the URLs that a browser accesses as a 2-mode graph $G = (U, V, E)$, where U are the URLs that the user explicitly visits, V are the URLs that are embedded within those pages, and E is the set of edges connecting vertices in U (explicitly visited URLs) to vertices in V (URLs embedded within visited pages). In this paper, we call vertices in U *referers*, vertices in V *hosts*, and G the *referer-hosts graph*.

In graph analysis, *communities* are groups of vertices that are well-connected internally, and sparsely connected with other groups of vertices. Vertices belonging to the same community are more likely to be similar with respect to connectivity and network position than vertices belonging to different communities. V contains both regular embedded objects and third-party objects potentially associated with trackers. We expect regular embedded objects to only appear on the hosting web page, while tracker objects need to appear on as many web pages as possible to enable successful tracking of users across websites. This implies that: 1) tracker vertices in V should be linked to many different vertices in U and 2) tracker vertices are members of well-defined communities in G.

Unfortunately, working with communities in 2-mode graphs like ours can be tricky. For example, the relationships between vertices in the same mode are only *inferred* from relationships that pass *through* vertices in the second mode, which can lead to unexpected results from standard community detection algorithms run on a raw 2-mode graph. This is especially a problem when the community

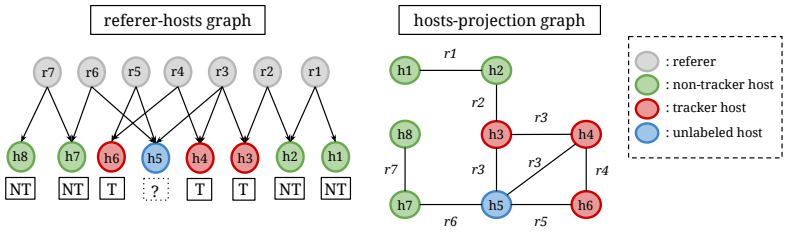


Fig. 1: Example of the hosts-projection graph transformation. Vertices prefixed with r are the pages the user explicitly visited while those prefixed with h were embedded within the r vertex they have an edge with. Note that additional information associated with the vertex (e.g., tracker/non-tracker/unknown label) is not affected by the transformation.

structures of the two modes are different as we might expect in our case [9]. To avoid this problem, it is typical to extract and analyze 1-mode projections of 2-mode graphs.

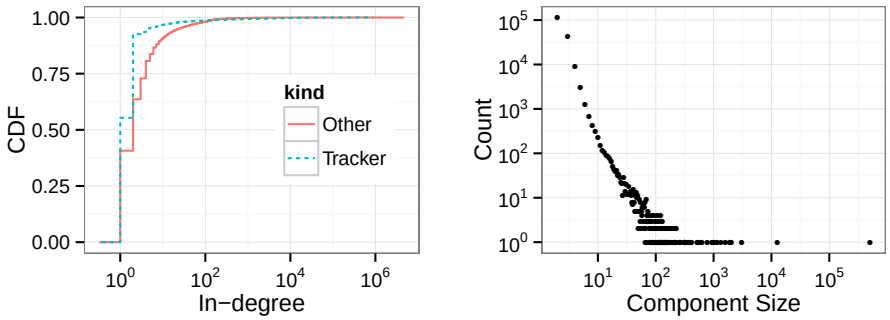
Assuming that users do not intentionally visit tracker sites, U should not contain tracker URLs which are instead contained in V. Accordingly, we can project the 2-mode graph into a 1-mode graph that only contains the vertices in V, by creating the *hosts-projection graph* G'. In G', we create an edge between any two vertices in V that share a common neighbor in G. I.e., if two vertices, v and v' from V both share an edge with a vertex u from U, then there is an edge $e = (v, v')$ in G'. Fig. 1 illustrates this transformation. This way, G' preserves much of the original graph's structural information and captures implicit connections between trackers through other sites.

3 Trackers' Position in the Graph

In this section we present an analysis on the referer-hosts graph and the hosts-projection graph. We are especially interested in discovering whether trackers have different properties than "normal" URLs in these graphs.

3.1 In the Referer-Hosts Graph

We first investigate trackers' degree centrality, or how well trackers are connected to other vertices in the graph. Although trackers are essentially required to appear on many different pages to collect meaningful data, we are interested in quantifying this. We begin by plotting the in-degree of vertices in mode V of the referer-hosts graph, broken down into "trackers" and "others" in Fig. 2a. The figure can be thought of as illustrating the number of unique referers that tracker/non-tracker hosts are embedded within. Surprisingly, we find that trackers tend to have a slightly lower in-degree than other URLs, which contradicts our initial observation that trackers must appear on many different pages in order to work. When looking at things a bit closer, we discovered that this is



(a) CDF of in-degree.

(b) Connected components distribution.

Fig. 2: Basic analysis of referer-hosts graph.

due to the use of *user/page specific tracking URLs*, mostly from Google, such `unique-hash.metrics.gstatic.com`. It follows that simply assuming high in-degree vertices as characteristic of trackers is not suitable. As we will discuss later, the hosts-projection graph transformation can be used to shed additional light on this situation.

Next, to see how well connected trackers are *to each other* we extract connected components and plot the distribution of their sizes in Fig. 2b. A connected component is a subgraph in which there exists a path between any two of its vertices. As expected, there are many 2-vertex components (pages that were only visited once and that host no, or very uncommon 3rd party content) and one dominant component. This largest connected component (LCC) contains 500,000 vertices, *i.e.,* one fourth of the distinct URLs in our dataset, and 94% of all trackers in our dataset (identified via EasyPrivacy list) are in the LCC. We will leverage this finding in Section 4 when showing how the community structure of trackers can be exploited for detection purposes.

3.2 In the Hosts-Projection Graph

We create the hosts-projection graph from the largest connected component in the referer-hosts graph. The projection has 80,000 vertices and 43 million edges. We note that the only substantive information lost in the projection graph is the number of unique pages a tracker appears on (i.e., the in-degree of the vertex within the referer-hosts graph). We first look at the degree distribution of trackers, and then examine the composition of their neighborhoods within the hosts-projection graph.

Trackers' degree distribution is a distinguishing factor Fig. 3 shows the degree distribution of trackers and other hosts in the hosts-projection graph. As opposed to the referer-hosts case, here we observe a clear difference between the degree distribution of trackers and other pages, noting that the low-degree skew

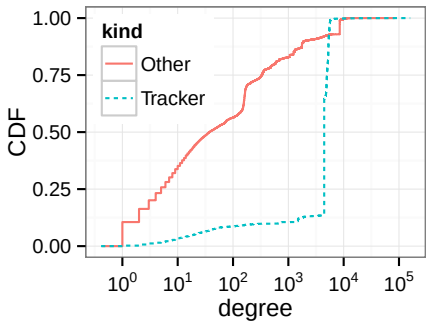


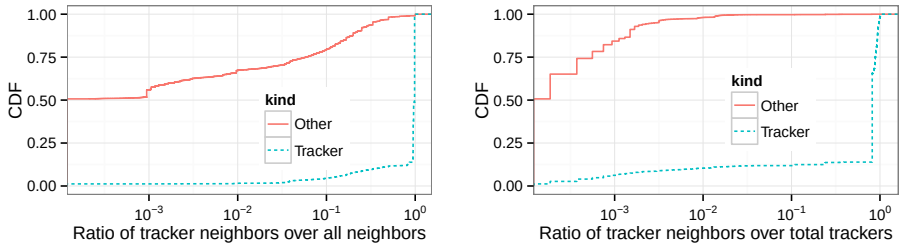
Fig. 3: CDF of degrees in the hosts-projection graph for trackers and others.

of trackers has disappeared. This is due to the construction of the projection graph: while in the referer-hosts graph, trackers only have edges to the pages they are embedded within, in the projection graph, they are directly connected with any URL that co-appears on the same page. For example, if we have three trackers that are only embedded within a single page, they will each have an in-degree of 1 in the referer-hosts graph, however, they will all be connected in the projection graph, resulting in a degree of 2. In general, we note that a higher degree might imply that trackers are more "important" in the projected graph than other pages.

Fig. 3 also illustrates another distinguishing factor of trackers. Their degree distribution skews extremely high, with about 80% having a degree of over 3,000. The explanation for this is that URLs that point to content on sites (e.g., CDNs) tend to be unique, or at least tend to appear on only a few sites. On the other hand, trackers *must* appear on multiple sites to be effective, and thus co-appear with many other URLs (some tracker, some not).

Trackers are mainly connected to other Trackers Next, we examine trackers' neighborhoods more closely. Fig. 4a shows the ratio of a vertex's neighbors that are trackers, distinguishing between tracker vertices and other. We observe that the vast majority of trackers' neighbors are other trackers. To further investigate how well-connected trackers are among them, we plot the ratio of a vertex's neighbors that are trackers over the total number of trackers in Fig. 4b and observe that *trackers tend to be direct neighbors with most of the other trackers in the graph.* This result is likely an artifact of publishers' tendency to add multiple trackers on their websites in the hope of serving better targeted ads.

From a privacy standpoint, this result is worrying as it highlights the pervasiveness of passive "surveillance" on the web. Even completely blocking any particular tracker could be somewhat mitigated by collusion. We do note that because the hosts-projection graph flattens the referer-hosts graph, collusion would not be enough to regain information on visits to pages where it is uniquely present.



(a) Ratio of the projection graph vertices' neighbors that are trackers.

(b) Ratio of a node's tracker neighbors over the total number of trackers in the dataset.

Fig. 4: CDFs of neighborhood compositions for trackers and non-trackers.

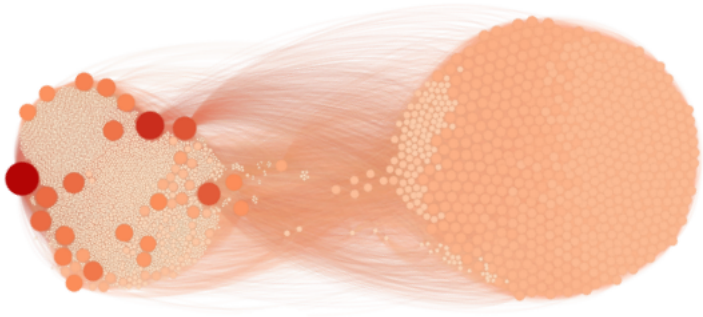


Fig. 5: Tracker oriented visualization of the hosts-projection graph from April's logs. The visualization includes only edges where at least one end point is a tracker, resulting in 60k vertices and 340k edges. The darker a vertex's color, the higher its degree. The community on the right contains trackers and ad servers, where ad servers can be seen as having a slightly lighter color and being mostly clustered on the left edge of the community. The left cluster consists of normal webpages and a few popular trackers, distinguished by their larger size and darker color.

Our findings up until now suggest that trackers form a dense community in the hosts-projection graph. This dense community is quite clearly seen in Fig. 5, which visualizes the host-projection graph (from April's logs) focused around trackers' positions. We observe that the majority of low-degree trackers indeed form a very dense and easily identifiable community (the cluster on the right). On the other hand, there exist a few *popular* trackers (the large dark nodes in the left cluster), which are connected to the majority of normal URLs and are also very well-connected among each other.

	Test Records in LCC	Trackers in LCC	Total New Trackers
Feb	13685	760	811
March	18313	740	774
April	40465	747	792

Table 1: New Trackers per Month

4 Classifying Trackers

Our findings suggest that trackers form a well-connected cluster in the hosts-projection graph, and are mostly connected to other trackers. In this section, we leverage these findings to automatically classify trackers. We show that even a simple assessment of vertices' neighbors in the hosts-projection graph can yield good classification results with two methods: 1) a rule-based classifier which analyzes the first-hop neighborhoods of each unlabeled vertex in the hosts-projection graph, and 2) an iterative label propagation method.

4.1 Classification via Neighborhood Analysis

This classification method analyzes the first-hop neighborhoods of each unlabeled node in the hosts-projection graph. For each unlabeled node, we count the number of trackers among its immediate neighbors and make a classification decision based on a configurable threshold. If the percentage of tracker neighbors is above the threshold, then the node is labeled as a tracker.

We evaluate our classifier using three subsets of our dataset. For every subset, we use all the hosts that appear in the last month as the test set and all the previous months as the training set. Thus, we use, e.g., the logs from November up to January in order to classify hosts seen in February logs. Hosts in the training set are labeled as "tracker" or "other" using the EasyPrivacy list as ground truth. Note that we also ensure that previously labeled vertices are not included in any future test sets. We use our classifier to tag each of the untagged nodes as *tracker* or *non-tracker* and measure precision, accuracy, false positive rate (FPR) and recall for each method. We assess classification stability, by randomly choosing test sets out of the complete dataset.

The number of test records and previously unseen trackers per month are shown in Table 1. We observe around 800 new trackers per month and this number is independent from the total number of requests to new pages over the 6 months of our dataset[3]. This indicates that there is enough diversity in the tracker "ecosystem" that users are constantly exposed to "new-to-them" trackers. In turn, this strengthens the need for an automated detection system to alleviate the load on crowdsourced approaches.

The classification results for February, March, and April are shown in Fig. 6. We assess the impact of threshold selection in the following ways: (a) Unlabeled

[3] We consider a tracker new if our users have not been exposed to it before. Note that we identify trackers by their unique URLs, without grouping them by domain.

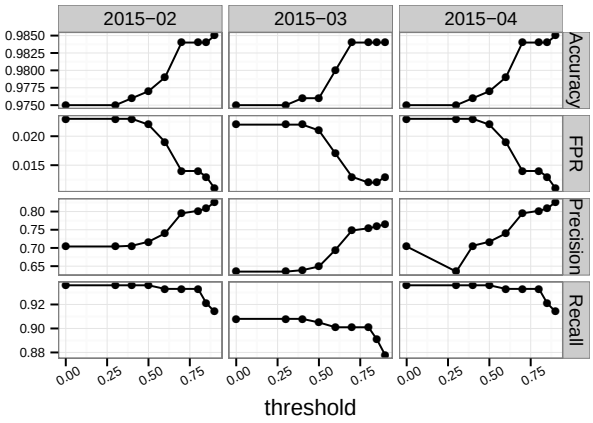


Fig. 6: Classifier performance for the neighborhood analysis method.

nodes take the label of their neighbors' most common tag (threshold = 0.00), and (b) The tag appears on at least a given fraction of the vertex's neighbors.

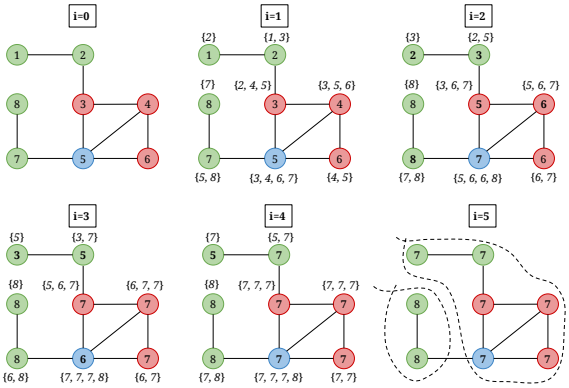
In all cases, we achieve a classification precision that varies from 64% up to 83%. We observe that precision increases for higher thresholds: the more tracker neighbors a node has, the higher the probability that it is a tracker itself. Similarly, FPR and accuracy both improve for higher thresholds, but remain under 2% and over 97% in all cases. On the other hand, recall decreases as we increase the threshold, which means that we might miss a few trackers, but it is above 88% in all cases.

4.2 Classification via Label Propagation

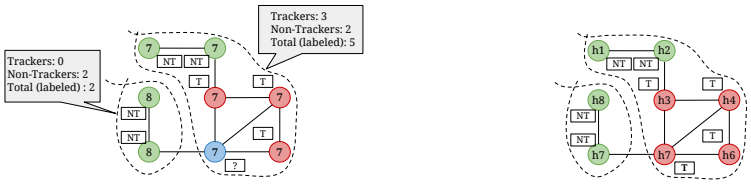
Label Propagation is a scalable iterative algorithm for community detection [11]. It exploits the graph structure to propagate labels and identify densely connected groups of vertices. Initially, vertices are assigned unique labels. Then, in an iterative fashion, vertices exchange labels with their neighbors. At each iteration, a vertex receives a list of labels of its immediate neighbors, adopting the most frequent label for itself. The algorithm converges when an iteration results in no label changes.

Fig. 7 illustrates how we use this algorithm on the hosts-projection graph. First, vertices propagate labels until convergence (i=0:4 in Fig. 7a); next vertices with the same label are grouped in the same community (i=5 in Fig. 7a). Then, we use the EasyPrivacy list to identify and *tag* known trackers inside the clusters, and tag as non-trackers white-listed vertices (Figure 7b). Finally, we assign a super tag to each cluster by choosing the most popular tag among its cluster members (Figure 7c). We classify unlabeled nodes by assigning them the super tag of the cluster in which they belong.

The results for the label propagation method are shown in Table 2. To assess classification stability, we evaluate the classification using random sets of test



(a) Cluster identification with label propagation.



(b) Assigning tags to individual vertices. (c) Super tag assignment to communities.

Fig. 7: Illustration of label propagation technique for tracker classification.

		precision	FPR	accuracy	recall
Monthly Test Sets	**Feb**	0.934	0.004	0.993	0.932
	March	0.946	0.002	0.994	0.9
	April	0.922	0.001	0.997	0.872
Random Test Sets	**5%**	0.923	0.004	0.994	0.958
	10%	0.934	0.004	0.993	0.941
	20%	0.941	0.003	0.994	0.948
	30%	0.939	0.003	0.994	0.951

Table 2: Label Propagation Classification Results

records of varying sizes. Instead of selecting the test set based on the timestamp of the log record, we create test sets from the complete graph, by randomly choosing log records and marking them as "untagged". We run this experiment for test sets of 5%, 10% , 20% and 30% of the complete dataset and repeat it 3 times.

By exploring further than the first-hop neighborhood of nodes, this method can successfully locate the trackers community and classify test items with extremely high precision, up to 94%, in addition to achieving high accuracy and recall, and lowering FPR. Further, this result does not come at a performance cost: the algorithm converged in less than 10 iterations for all the test sets used. Finally, this method has the advantage of not needing a manually set threshold.

5 Related Work

A number of studies have empirically analyzed the ecosystem of trackers and third-parties on the web, focusing on behavioral and network aspects.

TrackAdvisor [8] is a tool that analyzes cookie exchange statistics from HTTP requests to automatically detect trackers. Similar to us, their goal is to identify trackers without having to rely on blacklists. They also identify third-party requests by looking at the referer field. Their dataset is created by visiting Alexa top 10K pages (not real user data) and is an order of magnitude smaller than ours (500k requests in total). Our method does not need to intercept cookie traffic. Our finding that a tracker appears in multiple pages agrees with their results. In conclusion, we could call the two methods complementary; they could be combined to produce a more powerful tool.

Roesner et al. provide a study of web tracking, classifying tracking behaviors and evaluating defense mechanisms [13] using web traces from AOL search logs to simulate real user behavior. They build a classification framework for distinguishing different types of trackers, based on their functionality. In contrast, our classification method distinguishes between trackers and non-trackers, while it is oblivious to the tracker mechanisms and functionality specifics.

Bau et al. propose a machine learning mechanism for detecting trackers [4]. They evaluate machine learning approaches and present results from a prototype implementation. They use DOM-like hierarchies from the crawl data HTTP content headers as the main features. While they achieve precision of 54% for 1% FPR, our methods achieve much better precision and lower FRP, while not relying on page content collection.

Gomer et al. investigate the graph structure of third-party tracking domains in [6] in the context of search engine results. They obtain their graph by using searching several popular search engines with a set of pre-defined queries as opposed to real browsing behavior as we do. Their focus is on how users are exposed to trackers via normal search behavior and they find a 99.5% chance of being tracked by all major trackers within 30 clicks on search results. They further found that the graph structure was similar across geographic regions, which reduces the concern of bias in our dataset.

In agreement with our findings, most of the above works also find that a small number of trackers are able to capture the majority of user behavior. However, our work is, to the best of our knowledge, the first to show that using this community structure as an explicit feature can accurately predict whether an unknown URL is a tracker or not.

6 Conclusion

In this paper we explored the community structure of trackers using a large-scale dataset from an explicit web proxy. We transformed user requests into a 2-mode referer-hosts graph where vertices in the first mode represent pages the user visited and vertices in the second mode represent requests for objects embedded in those pages. We found that 94% of trackers were in the largest connected component of this graph. In order to study how trackers relate to each other, we collapsed the referer-hosts graph into a 1-mode hosts-projection

graph. From the hosts-projection graph we observed an extremely high degree of clustering, indicating the formation of tight communities. From this observation, we demonstrated the effectiveness of two tracker detection mechanisms: 1) a simple threshold based classifier that examines the number of tracker neighbors of unknown vertices and 2) a label propagation algorithm that makes implicit use of the communities trackers form. Both techniques achieved highly surprising accuracies (over 97%) and low false positive rates (under 2%).

We implemented the analysis and classification algorithms using Apache Flink [2]. In the future we intend to port them to a streaming version for deployment within our explicit web proxy, but even our initial implementations are quite fast. For example classification via the label propagation method was on the order of minutes when run on a commodity Mac laptop, indicating there are few scalability concerns for production deployment.

References

1. AdBlock. https://getadblock.com/.
2. Apache Flink. http://www.flink.apache.org.
3. EasyPrivacy List. https://hg.adblockplus.org/easylist/.
4. J. Bau, J. Mayer, H. Paskov, and J. C. Mitchell. A promising direction for web tracking countermeasures. In *Web 2.0 Security and Privacy*, 2013.
5. S. Englehardt, D. Reisman, C. Eubank, P. Zimmerman, J. Mayer, A. Narayanan, and E. W. Felten. Cookies that give you away: The surveillance implications of web tracking. In *Proceedings of the 24th international conference on World Wide Web*, WWW '15, 2015.
6. R. Gomer, E. M. Rodrigues, N. Milic-Frayling, and M. C. Schraefel. Network analysis of third party tracking: User exposure to tracking cookies through search. In *Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies*, pages 549–556, 2013.
7. B. Krishnamurthy and C. Wills. Privacy diffusion on the web: A longitudinal perspective. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 541–550.
8. T.-C. Li, H. Hang, M. Faloutsos, and P. Efstathopoulos. Trackadvisor: Taking back browsing privacy from third-party trackers. In *Passive and Active Measurement*, pages 277–289. Springer, 2015.
9. D. Melamed. Community structures in bipartite networks: A dual-projection approach. *PLoS ONE*, 9(5):e97823, 05 2014.
10. F. Papaodyssefs, C. Iordanou, J. Blackburn, N. Laoutaris, and K. Papagiannaki. Web identity translator: Behavioral advertising and identity privacy with WIT. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks (to appear)*, HotNets '15, 2011.
11. U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, 2007.
12. J. D. Rockefeller. Do-Not-Track Online Act of 2013. US Senate, 2013.
13. F. Roesner, T. Kohno, and D. Wetherall. Detecting and defending against third-party tracking on the web. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI '12, 2012.
14. O. Williams. Adblock extension with 40 million users sells to mystery buyer, refuses to name new owner. http://thenextweb.com/apps/2015/10/02/trust-us-we-block-ads/, 2015.