CLUE: Clustering for Mining Web URLs

CLUE: Clustering for Mining Web URLs

Andrea Morichetta, Enrico Bocchi, Hassan Metwalley, Marco Mellia
Politecnico di Torino
name.surname@polito.it

***Abstract*—The Internet has witnessed the proliferation of applications and services that rely on HTTP as application protocol. Users play games, read emails, watch videos, chat and access web pages using their PC, which in turn downloads tens or hundreds of URLs to fetch all the objects needed to display the requested content. As result, billions of URLs are observed in the network. When monitoring the traffic, thus, it is becoming more and more important to have methodologies and tools that allow one to dig into this data and extract useful information.**

In this paper, we present CLUE, Clustering for URL Exploration, a methodology that leverages clustering algorithms, i.e., unsupervised techniques developed in the data mining field to extract knowledge from passive observation of URLs carried by the network. This is a challenging problem given the unstructured format of URLs, which, being strings, call for specialized approaches. Inspired by text-mining algorithms, we introduce the concept of URL-distance and use it to compose clusters of URLs using the well-known DBSCAN algorithm.

Experiments on actual datasets show encouraging results. Well-separated and consistent clusters emerge and allow us to identify, e.g., malicious traffic, advertising services, and third-party tracking systems. In a nutshell, our clustering algorithm offers the means to get insights on the data carried by the network, with applications in the security or privacy protection fields.

I. Introduction

The web has become the most popular application of the modern Internet. Originally born to access hypertext, nowadays it is used to watch videos, play games, read emails, chat online, etc. Web pages have become much more complex, with dynamic elements offering personalized views, and are now rich of multimedia contents and web applications that run inside the browser. HTTP is the de-facto standard application-layer protocol [1], allowing the browser to retrieve the hundreds of objects composing a page with a simple request-response mechanism. Billions of objects are available on the web, each of them being identified by a Uniform Resource Locator (URL). Static URLs directly point to an object, e.g., portions of text or an image file like http://acme.com/index.html, but more and more frequently URLs encode queries that servers process to return a dynamic result. For instance, a Google search, a click on "like" buttons, or the images served by an advertisement platform are typical examples of dynamic URLs, e.g., http://acme.com/s?key=like.

Given the amount of URLs that are retrieved to fulfil ordinary browsing activities, monitoring and understanding the dynamics of the network is not an easy task. Due to the volumes of today's Internet and the complexity of its architecture, where resources are retrieved from remote servers, cloud data-centers or Content Delivery Networks (CDN), the work of network and security analysts requires advanced tools to effectively dig into such huge amount of raw data. Moreover, in many scenarios it is required to extract knowledge from it, e.g., for investigating an incident, or to extract signatures for Intrusion Detection Systems (IDS).

In this paper, we focus on the problem of automatically analyzing web traffic leveraging URLs. We design an unsupervised methodology that groups URLs in clusters according to a similarity metric. We call it CLUE, Clustering for URL Exploration. The goal is twofold. First, we reduce the number of items the analyst has to visualize and process, from hundred thousands of single URLs to few hundreds of clusters. Second, we target the identification of automatically generated URLs, e.g., URLs generated by advertisement platforms, polymorphic malware, or wiki-like systems.

Previous studies faced the analysis of URLs or entire web pages typically with a specific goal in mind, e.g., detecting duplicated pages, improving page rank, or pinpointing phishing websites – see Sec. II for a thorough discussion of related work. Our work differs from previous approaches as we aim at offering data exploration tools not tailored to a specific goal. We assume the system is fed by a set of URLs collected by passively observing HTTP requests produced by hosts in a live network. We specifically do not want to identify clients or keep track of their past navigation history, thus preserving users' privacy. In addition, we want to avoid the overhead introduced by web crawling techniques.

Ingenuity is required to design such a system. Unsupervised machine learning approaches like clustering algorithms have gained popularity. Clustering [2], [3] is defined as the task of grouping samples according to their similarity. Close samples are placed in the same cluster, while samples belonging to two clusters are far apart. Similarity is classically measured as the distance between two samples in a metric space, where the triangular inequality holds. The definition of a distance is unfortunately not trivial for URLs, which are indeed textual strings. We solve this problem proposing the *URL-distance*, a modification of the Levenshtein distance, which specifically takes into account key URL characteristics, i.e., string length and character frequencies which are different than in regular text strings. Next, we use DBSCAN, a well-known density-based clustering algorithm. As result, URLs are grouped into well-separated and cohesive clusters.

We assess CLUE performance considering a dataset of URLs accessed by ordinary users through PCs, tablets and smartphones. Half of the users are infected by a well-known polymorphic malicious software called TidServ [4], whose traffic is identified by an IDS. We use this labeled dataset as

This work has been funded by the WWTF Agency through the BigDAMA project.

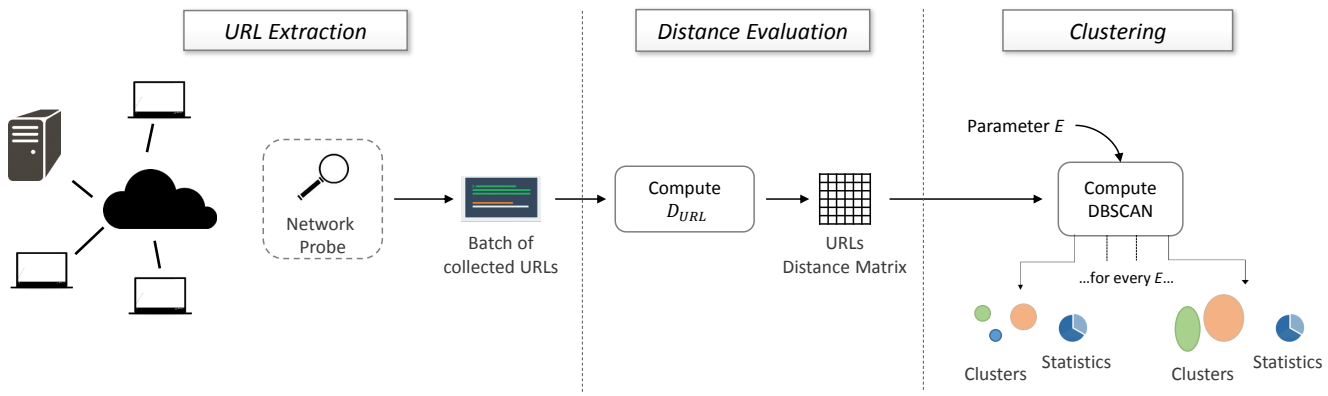


Fig. 1: CLUE system architecture overview.
Left to right, the three processing stages allowing us to group URLs in clusters: (i) URLs are extracted from a live network by means of a passive probe and collected in batch; (ii) the distance between each pair of URLs is computed through d_{URL} metric; (iii) DBSCAN clustering algorithm is applied. Additional statistics are complementary provided for the analyst.

ground-truth to tune parameters so that clusters of malicious URLs emerge. We then run the clustering algorithm on a larger dataset where no infected hosts are considered to show the potential of the approach. Results are encouraging: CLUE is able to pinpoint clusters of URLs generated by video streaming or advertisement services, and by malware families or third party tracking systems. A simple manual investigation is sufficient for the analyst to tie clusters to their category and augment the understanding of some phenomena. This strengthens the potential of CLUE to support the mining of URLs and of web traffic with applications to security and privacy protection fields. In a nutshell CLUE offers the analyst the chance to mine web traffic presenting clues about traffic and services.

This paper is organized as follows. We first discuss related works in Sec. II. Methodology is presented in Sec. III, where the CLUE is detailed. The dataset used for the experiments is described in Sec. IV, while results are discussed in Sec. V. At last, Sec. VI summarizes findings and comments on further potential improvements.

II. Related Work

Several papers in the literature aim at identifying similar web pages or URLs. Each work is targeting different problems or is tied to a specific application, with custom techniques being designed. The large majority of works look for structural features that help in distinguishing different classes of websites to consequently group or classify them. Such features can be referred (i) to the URL of a web page, here intended as sequence of characters; or (ii) to the payload of the page, consisting of its layout, formatting, and syntactical properties.

A group of previous works aims at clustering web pages directly using the text they contain. Such approach requires the complete retrieval of the page, and typically expensive text-processing algorithms. A notable example of clustering applied to web content is [5]. Authors propose a methodology to quantify the syntactic similarity between generic text files through the computation of resemblance and containment features. They apply such technique to 30 M documents retrieved from the web and run clustering algorithms on top. A similar and more recent approach is presented in [6], while [7] stresses the importance of algorithmic design to achieve high scalability of clustering algorithms.

In the context of web page clustering for specific applications, the authors of [8] apply clustering algorithms to disambiguate between people's name on the Web. They use a set of features coming both from the page content and from the URL. They split the URL into multiple components (e.g., domain name, path, parameters) and extract properties that have to be recombined together, making the whole process a thorough but expensive technique.

Authors of [9] give more importance to URLs rather than page content in the process of clustering websites. They propose a technique based on Minimum Description Length [10], which is applied to URLs. Additional features derived from content and structural properties are used only at a later, more fine-grained, clustering stage. [11] and [12] present rule-mining techniques applied on URLs only. The former is aimed at detecting web pages duplicates, while the latter presents an automated tool to explicitly detect malicious connections to Command and Control (C&C) servers through URL patterns. Only the detection of C&C is targeted.

Cantina [13] targets the automatic identification of phishing websites by analyzing URLs with text mining approaches. Authors of [14] target the same problem leveraging the Levenshtein distance to detect spelling mistakes that lead to phishing sites, while [15] applies clustering algorithms to identify spam campaigns from URLs posted on Facebook walls. Considering page ranking, [16] proposes a URL-based methodology to automatically spot "qualified" links, i.e., those implying merit of the targeted page, and noisy ones, e.g., advertisement and promotional links, that do not confer authority to a page.

Finally, authors of [17] and [18] propose web pages classification techniques solely based on websites URLs. Despite the goal of "classifying" a web resource goes beyond the scope of our work, both papers bring readers' attention to scalability issues and time requirements in case the actual content of a page has to be fetched, and to the feasibility of content analysis

when the semantics of a web page lie into images or graphical works and thus textual analysis is not applicable.

All previous proposals leverage some particular features in the structure of URLs they target and devise specific solutions to reach the goal. Our goal is instead to employ general-purpose data mining approaches to investigate URL structures and group together those URLs that look similar. In this respect, we aim at helping the analyst by sensibly reducing the amount of elements to analyze. We offer her the chance to check few hundreds of consistent URL groups, instead of several thousands of single URLs. CLUE is an exploration tool to dig inside the web.

III. CLUE SYSTEM DESCRIPTION

We aim at designing a completely unsupervised methodology that can support the work of a network or security analyst in extracting knowledge from the URLs the network carries. In this section, we first provide a description of the tools needed to identify and extract URLs from the network traffic. As second, we highlight the need to summarize the distance between URLs in a numeric fashion, considering several distance measures and their behavior in our field of application. Lastly, we guide readers through DBSCAN, a density-based clustering algorithm. Fig. 1 shows the overview of the CLUE architecture. It depicts the three macro processing stages and the components belonging to each of them. A detailed description of each step is proposed in the following.

A. URL Extraction

An URL is a reference to a resource which embeds two essential pieces of information: (i) The mechanism to retrieve the resource, i.e., the network protocol, and (ii) the location of the resource, i.e., the hosting server and the path to obtain it. In the web scenario, URLs point to hypertext, i.e., pages with text, images, and multimedia content. Being HTTP the de-facto standard application protocol [1], URLs now are used as identifiers to retrieve any type of content, from simple self-contained pages to rich personalized websites full of resources served by third party platforms, e.g., advertisements, video, social network plugins, etc.

The network carries billions of URLs and the understanding of the content being served through them is a complex task. URL parsing is nowadays much more complicated than in the past: the same resource can be retrieved from multiple nodes, e.g., any server replica in a CDN. Moreover, additional parameters are commonly used to fetch a specific object when multiple options are available, e.g., google.com/logo.png?xy=640x480 or google.co.uk/logo.png?xy=1024x768 may refer to the same resource. At last, automatic URLs are commonly found in Content Management Systems such as WiKi pages, in polymorphic malware, or in advertisement platforms. How to find similarities and offer the network or security analyst a scalable means to understand how the network is used is thus a challenging problem.

In the path from raw network traffic to URL clusters, the first step performed by CLUE is the extraction of URLs as they are requested by users. This can be done using logs readily available from proxy or firewall systems, or by extracting URLs directly from packets. In this work, we assume the latter case: a passive network probe is located on a link where it processes packets in real time. The probe extracts URLs and dumps them in batches for later post-processing. To do so, network flows carrying HTTP traffic have to be detected. Deep packet inspection techniques are employed for such purpose, i.e., to identify strings that match the syntax of HTTP GET or POST requests. When a HTTP request is found, the URL there contained is logged in a plain text file. When a batch of URLs is formed, it is finally possible to step ahead towards the computation of the distance among URLs. This task does not have real-time constraints and can be scheduled when the data collection from the network is complete or on demand too.

B. URL Distance Evaluation

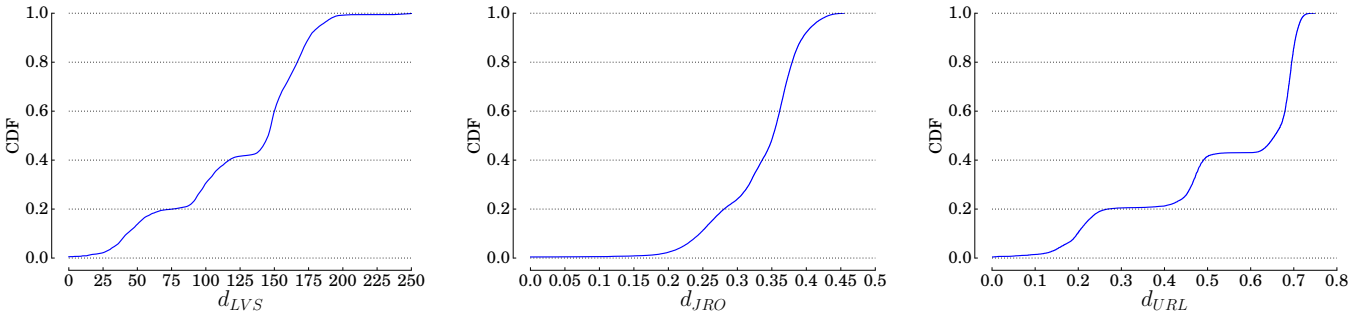
The concept of *distance* refers to a specific class of dissimilarity measures that aim at quantifying numerically the degree to which two points are far away [19]. A dissimilarity measure can be called distance if it meets three key properties that characterize a measure as metric: positivity, symmetry, and triangle inequality, respectively defined as

- $d(x_1, x_2) \geq 0,\ d(x_1, x_2) = 0 \iff x_1 = x_2$.
- $d(x_1, x_2) = d(x_2, x_1)$.
- $d(x_1, x_3) \leq d(x_1, x_2) + d(x_2, x_3)\ \forall x_1, x_2, x_3$.

The fulfillment of these properties is mandatory when dissimilarity measures are used on top of which data-mining techniques, clustering included, run. In our case, we look for a distance metric to compute the dissimilarity of strings. Distance measures suitable for application to textual strings take the name of "string metrics" or "string distance functions". The adoption of such metrics is popular in the field of text-mining but also in all the problems where it is required to compare groups elements for which one has no a-priori knowledge or understanding. Textual distance metrics therefore represent a convenient and viable way to compactly represent in numbers the dissimilarity among strings.

We focus on a particular class of distance metrics, the edit-distance based functions [20]. As the name suggests, the distance between two given strings s_1 and s_2 is intended as the minimum number of steps required to convert the string s_1 into s_2. Edit-distance functions have been used to target the analysis of free text where strings are well-formed words from a dictionary, with a defined grammatical syntax and with well-understood constraints.

The most popular technique is the *Levenshtein* distance [21] $d_{LVS}(s_1, s_2)$ that assigns a unitary cost for all editing operations, i.e., insert, remove, or replace one character. It computes an absolute distance between pairs of strings that is at most equal to the length of the longer string. This makes the Levenshtein distance inconvenient when comparing a short URL against a long one, as URL length possibly spans from few to hundreds of characters.



(a) d_{LVS} (b) d_{JRO} (c) d_{URL}

Fig. 2: CDFs of distances on the TidServ URLs.
To facilitate the consequent clustering of URLs, we aim at having distances concentrated in ranges. From the plot, it emerges that (i) d_{LVS} is potentially a good candidate but the outputted distance value is not weighted by the length of the strings being compared; (ii) d_{JRO} does not show any significant step in the distribution, posing serious issues in thresholds decision; (iii) d_{URL} appears to be the best candidate, showing three modes in its CDF.

The Levenshtein distance $d_{LVS}(s_1, s_2)$ is defined as

$$d_{LVS}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0. \\ min \begin{cases} d_{LVS}(i-1,j)+1 \\ d_{LVS}(i,j-1)+1 \\ d_{LVS}(i-1,j-1)+I(s_{1i} \neq s_{2j}) \end{cases} & \text{otherwise.} \end{cases}$$

where $d_{LVS}(i,j)$ is the distance between the first i characters of s_i and the first j characters of s_2. I is the indicator function, namely equal to 0 when $s_{1i} = s_{2j}$.

A different approach is taken by the Jaro distance. In this case, the distance function considers the number and the order of common characters between two strings. Let m be the number of matching characters, and t be half the number of transpositions. The Jaro distance $d_{JRO}(s_1, s_2)$ is defined as

$$d_{JRO} = \begin{cases} 1 & \text{if } m = 0. \\ 1 - \frac{1}{3}(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}) & \text{otherwise.} \end{cases}$$

Given the peculiarity of URLs, whose length may vary widely and which may include random substrings, we propose a custom modification of the Levenshtein distance, d_{LVS2}. Specifically, we count the total number of insertions and deletions, but we weight replacement by a factor of two. The rationale is that a replacement corresponds to one combined operation of deletion and insertion. We also explicitly consider the string length, and normalize the results in a $[0, 1]$ range as follows:

$$d_{URL}(s_1, s_2) = 1 - \frac{|s_1| + |s_2| - d_{LVS2}(s_1, s_2)}{|s_1| + |s_2|}$$

This leads to a bounded distance metric, and specifically $d_{URL} = 0$ if $s_1 = s_2$, while $d_{URL} = 1$ if the two strings are completely different.

To give the intuition of the different results achievable, consider a simple example. Let s_1 be "google.com" and s_2 be "1goggle.com". We now compute the numerical value provided by each of the considered distance functions. The Levenshtein distance $d_{LVS}(s_1, s_2) = 2$, accounting for one insertion ("1") and one replacement operation ("o" $\rightarrow$ "g"). For d_{JRO}, the number of matches m is 9 (g,o,g,l,e,.,c,o,m), and the number of transpositions t is 0. Thus $d_{JRO} = 0.094$. Finally, $d_{URL} = 0.143$ since we have one insertion, weighted 1, and one replacement, weighted 2.

We now run a simple experiment to raise awareness on the importance of choosing an adequate distance function. We consider all the URLs found in our dataset (see Sec. IV for details) that have been generated by TidServ, a polymorphic malware. We then compute the distance between any pair of URLs (u_1, u_2) according to the different definitions of $d(u_1, u_2)$ reported above. Fig. 2 shows the Cumulative Distribution Function (CDF) of the measured distances for d_{LVS}, d_{JRO}, and d_{URL}, respectively.

Given our goal is to cluster elements that are "close" one to the other, we prefer to have distances concentrated in ranges. A pair of similar elements should exhibit a small distance, while a pair of different elements should exhibit a very large distance. d_{LVS} shows three groups in its CDF, suggesting for potential clusters. However, d_{LVS} support is not bounded in a given range (in our experiments, it spans in the [0:250] range), since no normalization is entailed. This makes the comparison mostly driven by string lengths, i.e., any two short strings will be much more similar than any two long strings. d_{JRO} instead results in a nearly-continuous shape, showing no clear steps that would help in separating close from far away pairs. d_{URL} satisfies the intuition of having distance ranges, as it clearly shows three modes in the CDF. Moreover, its support is bounded in the [0:1] range, normalizing the distance with respect to the length of the two considered strings.

In our methodology, we compute the distance by submitting the entire URL as a single string made by hostname and path. We reached this decision after performing some experiments, not reported here for brevity, where we tested several definitions of d_{URL} by considering hostname and path separately, and then by composing a single metric via linear combinations. However, blending hostname and path distance values resulted not straight forward and did not lead to better results.

C. Clustering

Clustering algorithms are unsupervised machine learning approaches that aim at grouping together points according to similarity metrics [2], [3]. They offer exploratory means to analyze raw data, sensibly reducing the number of elements to be analyzed from hundred thousand individual points to few hundreds clusters. Elements that are grouped in the same cluster share common features and, as such, can be analyzed a single entity. The homogeneity of items in a cluster allows the analyst to naturally extract knowledge about the elements themselves. Clustering algorithms are thus good candidates to process URL distances and group together those URLs that show a low distance value.

Among clustering algorithms, density-based approaches define clusters as the set of elements that form areas of higher density than the remainder of the data set. Such techniques have several notable and useful advantages: (i) They do not require any knowledge on the final number of clusters in advance (one of the major weaknesses of centroid based clustering approaches, as *k-means*); (ii) they can find arbitrarily shaped clusters; (iii) they include the notion of outliers, which are left unclustered as noise. We rely on DBSCAN [2], one of the most popular density-based algorithms.

To better illustrate how density-based clustering algorithms work, consider a set of points in a sample space to be clustered. Let $d(x_1, x_2)$ be the distance between two points x_1 and x_2. Consider now the sphere of radius E centered in x_1. If at least $minPoints$ are within distance E from x_1, the point x_1 is classified as "core point". Formally, a given point x_1 is a core point if at least $minPoints$ are within distance E from it. These points are defined as "directly reachable" from x_1. A generic point x_k is "reachable" from x_1 if there exists a path $x_1, x_2, \ldots, x_k$ so that x_{i+1} is directly reachable from x_i. Reachable points from x_1 form a cluster, i.e., a dense region. Points that are not reachable from x_1 are called "outliers", and may either form a separate cluster if they belong to another dense region, or fall in the "noise" region if it is not the case.

$minPoints$ and E are two tunable parameters that can be set by a domain expert if the data to be processed is well understood. $minPoints$ defines the minimum size of a cluster and has little impact on the final results. E instead is a key parameter. If set too small, it leads to a high number of small clusters and lots of unclustered points. If set too large, it leads to few clusters with lots of (heterogeneous) points. Sensitivity analysis is thus essential to properly choose E. We better detail parameter choice impacts in Sec. V-B.

IV. Scenario and Dataset

In this section we provide an overview of the technologies used to record network traffic and of the tools used to extract useful information. We consider a scenario in which a sniffer passively monitors the traffic generated by a group of hosts, e.g., hosts in a LAN network, or households connected to a Point-of-Presence (PoP) of an Internet Service Provider (ISP). The sniffer is capable of identifying HTTP requests, and log them to a file for later postprocessing.

In our case, we capture traffic at the PoP of an European ISP where approximately 20,000 customers are connected. Most of them are residential customers accessing the Internet via ADSL modems. We instrument the PoP with a passive probe to monitor the traffic generated by residential users. The probe runs Tstat [22], a passive monitoring tool that rebuilds each TCP flow, tracks it, and, when the connection is closed, logs a *record* reporting statistics in a simple textual format. When the application protocol is HTTP, Tstat extracts the URL and logs it in file. In case multiple HTTP transactions are present due to the usage of HTTP-persistent option, multiple records are logged. We let Tstat collect URLs for an entire day, generating more than 100GB of data.

TABLE I: Dataset characteristics.

	All hosts	**TidServ infected hosts**
HTTP Flows	267,393	171,863
HTTP Volume	89.99 GB	44.16 GB
Total URL	411,727	255,304
Unique URL	78,421	43,479
Unique Tidserv URL	228	228

We have also access to a commercial Intrusion Detection System (IDS) that we use to label URLs as possibly malicious. The IDS has at its disposal an internal database of rules modeling network threats. If some URL matches one (or more) of these rules, the IDS raises an alert and flags the URL with a *Threat-ID*, i.e., a numeric code identifying a specific threat. For our purposes, we enabled signatures for a specific malware called *TidServ* (see Sec. V for a description of the malware) that is known to use polymorphic strings in the URLs to evade detection techniques. We identified 14 hosts to be infected by the malware in our dataset.

In the following, we consider one dataset. Table I provides some statistics about characteristics in terms of volume, number of URLs, etc. Our dataset considers traffic generated by the 14 hosts infected by the TidServ malware, i.e., for which the IDS flagged at least one flow as malicious, and 20 additional hosts randomly selected from the population of users; none of the URLs of this second group of hosts are flagged by the IDS. In total, more than 411,000 URLs are present, 78,421 of which are unique. For the sake of completeness, Table I details also the statistics considering only the 14 infected hosts. Out of the 255,000 total URLs, 43,479 are unique, of which only 228 are flagged as TidServ.

V. Results

In this section, we first present experiments to tune CLUE parameters. Then, we provide proofs about the CLUE effectiveness by presenting case studies and examples of analysis it enables.

A. TidServ overview

We firstly provide some highlights on the *TidServ* traffic. TidServ, also known as Alureon, TDSS or TDL [4], is a popular Trojan Horse. After infecting an host and transforming it in a bot, it communicates with a Command-and-Control

TABLE II: Examples of TidServ URLs flagged by the IDS. Common substrings in bold.

swltcho81.com/NZf4A07d7r7yE1C1dm**VyPTQu**MCZiaWQ9YjZjYWVhNjE0NjhhMmQ4ZTc0OGQ3ZTEzMTIyMDZiMDQ4NWY2MjJhYSZha**WQ9NDAxOTcmc2lkPTAmcmQ9MCZlbmc9d3d3Lmdvb2dsZS5pdCZxP**XVpbmZlIG15ZGVzaw==38c
rammyjuke.com/kaI1wWRd8Y5yfbU9dm**VyPTQu**MCZiaWQ9YjZjYWVhNjE0NjhhMmQ4ZTc0OGQ3ZTEzMTIyMDZiMDQ4NWY2MjJhYSZha**WQ9NDAxOTcmc2lkPTAmcmQ9MCZlbmc9d3d3Lmdvb2dsZS5pdCZxP**WZvcnVtIGFybWF0YSBkZWxsZSB0ZW5lYnJl37g
bangl24nj14.com/TVq2BttP743qt1c8dm**VyPTQu**MCZiaWQ9YjZjYWVhNjE0NjhhMmQ4ZTc0OGQ3ZTEzMTIyMDZiMDQ4NWY2MjJhYSZha**WQ9NDAxOTcmc2lkPTAmcmQ9MCZlbmc9d3d3Lmdvb2dsZS5pdCZxP**XVpbmZlIG15ZGVzaw==05c
iau71nag001.com/Kvb13nWd6P4XrFs3dm**VyPTQu**MiZiaWQ9MDU0NWQwZDQwY2MyODU4YWNjYzFlZjJkM2FiZDA5N2RiYmRlYmVkZiZha**WQ9N**TAwMTgmc2lkPTAmcmQ9MCZlbmc9d3d3Lmdvb2dsZS5pdCZxPWZhY2Vib29r27c

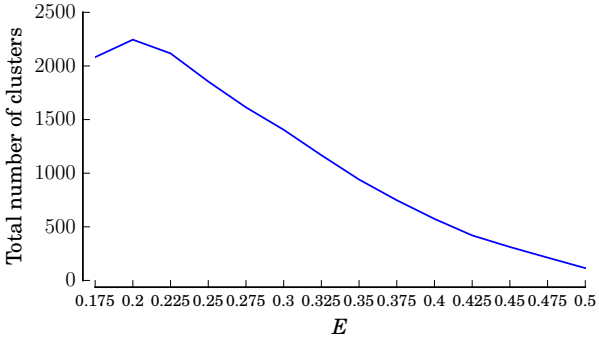
(C&C) server to receive commands.[1] C&C servers are typically contacted using HTTP to evade firewalls. Originally, static URLs were used, and security software could easily block the communications using, e.g., static rules and blacklists. However, recently malware started to evade IDS rules by using polymorphic approaches, e.g., randomly generating and rotating hostnames on the DNS for C&C servers, or adding randomness in the URL path. This makes more difficult to create static blacklists based on simple string matching.

TidServ adopts this expedient and changes periodically the URLs to contact C&C servers. To give the reader the intuition of how random can a TidServ URL appear, Table II reports four examples of URLs that the IDS flagged. Hostnames and paths change, but some common parts (in bold) are still visible (and could be used by the IDS to flag them). Sometimes the common pattern may be very long, but sometimes as few as 4 characters are found in common, suggesting different communication patterns may be present. Observing these patterns is easy if one is provided by the correct set of URLs. But finding them when mixed in the hundred thousands of URLs generated by a host makes the identification very challenging.

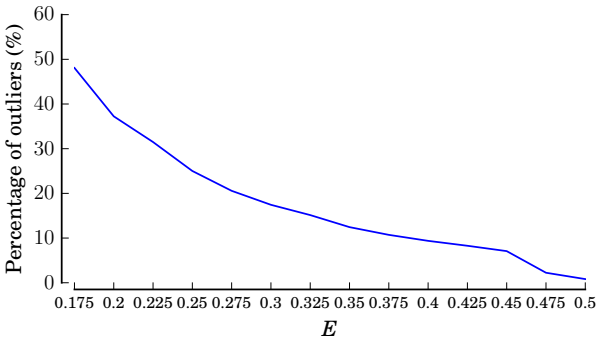
B. Parameter settings

We now run CLUE on the overall dataset. We fix $minPoints$ to 4 to let CLUE consider clusters of at least 4 points. This choice is not critical since normally one is interested in observing clusters with much higher number of points. The impact and choice of E needs a preliminary study since it can completely change clustering results. Intuitively, a too small value of E leads to a large number of very small clusters and to a lot of points that fall in a not-dense region, i.e., to a large number of outliers points. Conversely, a too large values of E leads to few and very big clusters, where all points are core points, sometimes artificially connected together by a single path.

[1]The C&C servers run on centrals computer that attackers use to update and control infected hosts.



(a) Number of clusters varying E.



(b) Percentage of outliers varying E.

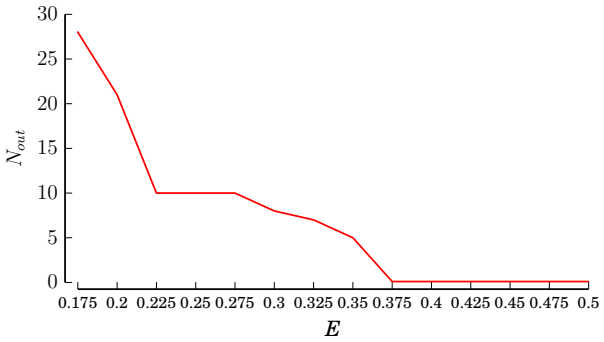
Fig. 3: Number of clusters and percentage of outliers versus E. Generic dataset.

Fig. 3 reports results when considering $E \in [0.175, 0.5]$. Recall $d_{URL} \leq 1$ by construction. Specifically, Fig. 3a and Fig. 3b report the total number of clusters and the percentage of outliers, respectively. As expected, both curves decrease as E increases. Since we are interested in some hundreds of clusters, these results suggest to use large values of E (between 0.4 and 0.45). Notice that for $E > 0.45$, a sudden drop in the number of outliers is observed due to the emerging of gigantic clusters.

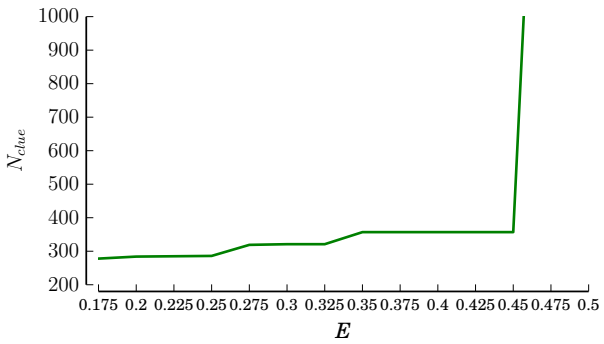
We now focus on how TidServ URLs are clustered. Intuitively, we would like to have them all clustered together, possibly in different clusters (cfr. Fig. 2), with none of them left in the noise region, and with only TidServ URLs being present in each cluster. We run CLUE for increasing values of E, and then we study how TidServ elements are clustered by leveraging the IDS labels as ground truth. We report results in Fig. 4. We consider two performance metrics:

(i) Number of TidServ in Outliers - N_{out}: the number of URLs labeled as malicious by the IDS which are left unclustered;

(ii) Total Clustered TidServ URL - N_{clue}: the total number of URLs belonging to any cluster in which there is at least a TidServ URL. This includes both TidServ labeled URLs, and eventual other URLs which the IDS did not label as malicious, but that CLUE included in the same cluster.



(a) N_{out} versus E.



(b) N_{clue} versus E.

Fig. 4: Number of TidServ elements left unclustered and total number of URLs in TidServ clusters.

Fig. 4a reports N_{out} versus E. If set too small, a large number of TidServ URLs results unclustered, since the randomness added in the URLs makes d_{URL} too large compared to E. Conversely, for values of $E \geq .375$, all TidServ labeled URLs are assigned to a cluster. Look now at Fig. 4b which depicts N_{clue}. We observe how more and more URLs gets clustered when E increases. Steps correspond to outliers being included into clusters. However, when $E > 0.45$, a sudden increase is observed, i.e., URLs suddenly gets merged in very large clusters, with $N_{clue} > 2600$ elements.

These results suggest that the best setting of E is in $[0.375, 0.45]$. We set it to 0.4 in the following. With this settings, CLUE isolates 574 clusters in total, 7352 outlier URLs on the whole dataset. All TidServ elements are assigned to a cluster. In particular, 7 clusters have at least 1 TidServ labeled URL. These clusters contain in total 357 URLs. However only 228 are labeled by the IDS, and we need to investigate if there are some not malicious URLs that are mistakenly clustered with TidServ URLs.

C. TidServ findings

Table III reports details for each of the 7 TidServ clusters. It reports the number of URLs, the number of labeled ones, the number of unique hostnames and the longest common substring for hostnames. Cluster are sorted by number of IDS labeled TidServ elements. The table shows the polymorphic behavior of the malware: several hostnames are used for the C&C server, and randomness is introduced in the URL paths too. CLUE is able to identify all TidServ labeled URLs in the total set of 78421 URLs. For instance, consider the first cluster. It is the largest one, with 192 URLs pointing to resources apparently hosted on 14 different hostnames, whose common pattern is just the .com substring. Interestingly, CLUE groups 118 different URLs that the IDS flags as TidServ, and 74 additional URLs which the IDS did not flag. By manually checking the latters, they present very strong similarities with the labeled URLs. Overall, CLUE associates to TidServ 357 unique URLs, of which only 228 are labeled by the IDS. With a manual validation, we verified that all the not-labeled elements are very likely malicious communications associated to TidServ. We hypothesize the IDS signatures do not cover them, and those can be considered false negatives. To give the reader some insight, Table IV reports some of the URLs of the last cluster in Table III. The IDS flags only 1 URL out of the 37 URLs CLUE clusters together. By looking at strings in Table IV, it is easy to spot some very strong similarity, hinting to all those URLs to be indeed TidServ related ones.

These promising results clearly show that CLUE can easily improve the analyst knowledge, finding new malicious patterns that could be used to improve IDS and firewall signatures.

TABLE III: TidServ clusters identified by CLUE. $E = 0.4$.

Tot. URL	TidServ URL	Hostnames	Hostname common string
192	118	14	.com
79	75	1	wuptywcj.cn
32	18	2	clickpixelabn.com
6	6	1	biiwf3iidpkxiwzqmj.com
6	5	1	zl091kha644.com
5	5	1	zhakazth.cn
37	**1**	**3**	**.com**

TABLE IV: TidServ URLs identified by CLUE as belonging to the same cluster. Only the first one was identified by the IDS. Common patterns are highlighted in bold.

gnu4oke0r.com/4VY00y9P7Z5xiPs9dmVyPTQuM**CZiaWQ9NWJjNWFiMjE1YjRmN2I4ZjM3OTRmODNkZjhmNWY0ZjFmODZkYjE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9M**CZlbmc9d3d3Lmdvb2dsZS5pdCZxPWxvdWlzIGNydWlzZXM=16h
lkckclcklii1i.com/sVV2mztX5W4xXZu5Y2xrPTIuN**CZiaWQ9NWJjNWFiMjE1YjRmN2I4ZjM3OTRmODNkZjhmNWY0ZjFmODZkYjE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9M**A==15g
lkckclckl1i1i.com/sVV2mztX5W4xXZu5Y2xrPTIuN**CZiaWQ9NWJjNWFiMjE1YjRmN2I4ZjM3OTRmODNkZjhmNWY0ZjFmODZkYjE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9M**A==15g
lkckclcklii1i.com/TAR3vUsX844qz1c5Y2xrPTIuN**CZiaWQ9NWJjNWFiMjE1YjRmN2I4ZjM3OTRmODNkZjhmNWY0ZjFmODZkYjE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9M**A==27g
lkckclckl1i1i.com/TAR3vUsX844qz1c5Y2xrPTIuN**CZiaWQ9NWJjNWFiMjE1YjRmN2I4ZjM3OTRmODNkZjhmNWY0ZjFmODZkYjE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9M**A==27g

D. Silhouette analysis

In this section, we present some interesting findings obtained by running CLUE on the whole dataset. We aim at exploring clusters, and observe if they provide some useful information. To select which cluster to analyze, we rely on the *silhouette analysis*, an unsupervised methodology to find how well each object lies within its cluster. Silhouette coefficient $s(i)$ measures how close a point i is assigned to its cluster. It computes the average distance $a(i)$ of i with all points in the same cluster; and $b(i)$ as the minimum average distance of i to points in other clusters. Then

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

where $a(i) = \sum_{j \in C} d_{URL}(i,j)$, C being the cluster i belongs to; $b(i) = min_{C' \neq C} \sum_{j \in C'} d_{URL}(i,j)$; it results $s(i) \in [-1, 1]$. Values close to 1 indicate that the sample is far away from the neighboring clusters, and it means that core point i is very close to all other core points in its cluster, i.e., cluster C is very compact. Instead, values close to 0 indicate that i is on or very close to the decision boundary between two neighboring clusters; finally, negative values indicate that those samples might have been assigned to the wrong cluster. The average $S(C) = E[s(i), i \in C]$ over all points in cluster C is a measure of how tightly grouped all the elements in the cluster are, i.e., how good is the cluster.

Since silhouette analysis permits to understand the quality of clustering, we firstly study the distribution of $S(C)$ among clusters. We consider only those clusters with more than 20 elements. We observe that 14% of clusters have a silhouette coefficient lower than 0.3, 37% between 0.3 and 0.5, 29% between 0.5 and 0.7 and, finally, 20% of clusters with a coefficient greater than 0.7. In a nutshell, most of the clusters have elements with a strong connectivity between them ($S(C) \geq 0.5$ for 49% of clusters), and only a small subset of clusters appears to be sparse. These results suggest that the choice of E and $MinPoints$ parameters are fitting to our case.

E. Generic findings

1) Mining strongly connected clusters: Silhouette analysis can easy the choice of which cluster to analyze. For instance, it suggests to the analyst those clusters whose points are very similar inside the cluster, and very different from points in the rest of clusters. Table V shows the effectiveness of CLUE in giving the analyst clues about the traffic generated by the system. Focus first on the top part of the table, which ranks clusters by decreasing $S(C)$. All these clusters but the TidServ one contains URLs sharing a single hostname with however different paths. The first cluster contains 551 unique elements. Manually checking those, it is straight forward to observe that all those are related to the Sky Go video streaming service, supported by Akamai CDN, using both HTTP Live Streaming (HLS) and Video on Demand, using MP4 and H.264 formats. The latter information is extracted from URL paths, not reported for brevity. Paths are different, but follow a clear syntax that allows CLUE to cluster them.

$S(C)$	Main hostname (unique number)	Elements	Activity
0.92	skygo_streaming-i.akamaihd.net (1)	551	Streaming
0.91	ad.doubleclick.net (1)	99	Advertising
0.87	cookex.amp.yahoo.com (1)	61	Malware
0.85	static.simply.com (1)	25	File Hosting
0.81	d24w6bsrhbeh9d.cloudfront.net (1)	63	File Hosting
0.81	mfdclk001.org (1)	27	Malware
0.78	adserver.webads.it (1)	35	Advertising
0.77	.com (3)	37	TidServ
0.75	pixel.quantserve.com (1)	57	Advertising
0.72	watson.microsoft.com (1)	29	Windows Debug
0.7	coadvertise.cubecdn.net (1)	36	Advertising
0.65	su.ff.avast.com (1)	82	Avast Update
0.64	log.dmtry.com (1)	24	Advertising
0.61	clickpixelabn.com (1)	32	Malware
-0.19	. (5468)	43592	Normal Traffic
0.26	fbcdn (22)	10572	Facebook CDN
–	Outliers (1783)	7352	–
0.61	feeds.wordpress.com (1)	1043	Feeds
0.69	atdmt.com (2)	768	Tracking
0.09	doubleclick.net (9)	691	Advertising
0.92	skygo_streaming-i.akamaihd.net (1)	551	Streaming
0.31	repubblica.it (6)	186	News
0.43	doubleclick.net (2)	171	Advertising
0.47	exch-eu.atdmt.com (1)	150	Tracking

TABLE V: Clusters information sorted by Silhouette coefficient on the top, and and for number of elements in the bottom. In all cases, clusters clearly pinpoint specific services.

Continuing the analysis, we observe some legitimate services related with advertisement platforms, all using explicit URL formats for parameter exchange. Identifying them is very easy, e.g., by performing a simple Google search [23]. Except these licit cases, the top clusters show many services with less legitimate purposes. Some services, e.g., cookex.amp.yahoo.com, mfdclk001.org clickpixelabn.com, are associated to malware activities. Also in this case, a simple Google search unveil immediately clues to understand the picture. The first cluster is associated to spyware actions[2]. Beside the common hostname, we find the common part "*http%3A//ad.yieldmanager.com/imp*" in the URL path. Probably this malware is connected to click fraud activities, and the C&C server hosted at *cookex.amp.yahoo.com* instruments the bots to perform fake-clicks on legitimate ads hosted on the Yieldmanager infrastructure. The second hostname is linked to a modified version of TidServ[3], and the last is a C&C server of another malware[4]. Also in this case, the analysis of URLs gives interesting hints about the possible malicious activities, not discussed for the sake of brevity. In all cases, the availability of several URLs easy the signature extraction job, e.g., to augment IDS or firewalls coverage.

2) Mining bigger clusters: The second part of Table V reports clusters sorted by number of unique URLs. The largest cluster contains many elements with a high number of different hostnames. It aggregates all "normal" traffic into the network, whose URLs are well-formed, with hostnames and paths that are syntactically simple. Given the very large number of services, it is possible to transform a URL into a similar one so that a large dense area is identified. Notice that this cluster is the only one with a bad Silhouette index. It would be appropriate to then run CLUE on these URLs with a smaller

[2]https://www.google.it/#q=cookex.amp.yahoo.com
[3]https://www.google.it/#q=mfdclk001.org
[4]https://www.google.it/#q=clickpixelabn.com

choice of E to better split URLs. We leave this for future work.

The second largest cluster aggregates URLs from the Facebook CDN, as easily highlighted by the common substring found in hostnames. It serves all images, videos, and static Facebook content. The third largest group of URLs collects all outliers. Here we find the 7252 URLs that fall outside any dense area. For these, it would be appropriate eventually to run CLUE with largest choice for E. Going down in the list, we observe URLs generated automatically by blog platforms, by advertising platforms, by newspapers websites, and by third-party tracking services [24]. These services follow users during their online activities identifying them using different techniques. User identifiers and other information about advertising (e.g., timestamp or banner size) are often sent to the server embedded into URL queries. Since these elements are obviously created artificially, and follow a regular syntax, they are easily identifiable by CLUE. This is why these services are so frequent in our results. By analysing the URLs, it is possible to unveil some of the mechanisms they use to track online activities. For instance, *doubleclick.net*, an advertising service subsidiary of Google, communicates using artificial URLs, with common parts, like "*u*=", "*kvid*=", "*kpid*=" or "*sz*=", that clearly carry information about user and her navigation. CLUE can thus also help researchers, network or security analysts to find (and eventually filter) these services.

In a nutshell, CLUE proves very useful in grouping similar URLs, thus easing the analysis of traffic, and naturally letting common patterns to emerge. Applications for security and privacy are straightforward.

VI. Conclusions and Future Work

In this paper we presented CLUE, an unsupervised system to mine URLs from passive traces. Based on unsupervised machine learning, it offers the analyst the ability to explore well defined clusters where similar URLs are grouped. Thanks to the cohesiveness of clusters, clues about traffic easily emerge and let the analyst to obtain information and identify possibly malicious traffic or advertisement and third-party services.

We presented results using a real but small dataset. We are now working on improve scalability of CLUE so it can work in real-time. As further improvement we are also working in the design of a hierarchical approach where large clusters can be further analysed by changing CLUE parameters.

References

[1] L. Popa, A. Ghodsi, and I. Stoica, "Http as the narrow waist of the future internet," in *ACM Hotnets*, 2010.

[2] C. C. Aggarwal and C. K. Reddy, *Data Clustering: Algorithms and Applications*. Chapman and Hall/CRC, 2013.

[3] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Pearson, 2005.

[4] Backdoor.Tidserv, https://www.symantec.com/security_response/writeup.jsp?docid=2008-091809-0911-99.

[5] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," *Computer Networks and ISDN Systems*, vol. 29, pp. 1157 – 1166, 1997.

[6] V. Crescenzi, P. Merialdo, and P. Missier, "Clustering web pages based on their structure," *Elsevier Data & Knowledge Engineering*, vol. 54, no. 3, pp. 279–299, 2005.

[7] T. Haveliwala, A. Gionis, and P. Indyk, "Scalable Techniques for Clustering the Web," in *International Workshop on the Web and Databases*, 2000.

[8] E. Elmacioglu, Y. F. Tan, S. Yan, M.-Y. Kan, and D. Lee, "Psnus: Web people name disambiguation by simple clustering with rich features," in *International Workshop on Semantic Evaluations*, 2007, pp. 268–271.

[9] L. Blanco, N. Dalvi, and A. Machanavajjhala, "Highly Efficient Algorithms for Structural Clustering of Large Websites," in *ACM WWW*, 2011.

[10] P. D. Grunwald, *The Minimum Description Length Principle*. The MIT Press, 2007.

[11] A. Agarwal, H. S. Koppula, K. P. Leela, K. P. Chitrapura, S. Garg, P. K. GM, C. Haty, A. Roy, and A. Sasturkar, "URL Normalization for De-duplication of Web Pages," in *ACM CIKM*, 2009.

[12] N. Kheir, G. Blanc, H. Debar, J. Garcia-Alfaro, and D. Yang, "Automated classification of C&C connections through malware URL clustering," in *Springer ICT Systems Security and Privacy Protection*, 2015, pp. 252–266.

[13] Y. Zhang, J. I. Hong, and L. F. Cranor, "Cantina: A content-based approach to detecting phishing web sites," in *ACM WWW*, 2007.

[14] M.-E. Maurer and L. Hofer, "Sophisticated phishers make more spelling mistakes: Using url similarity against phishing," in *Springer Cyberspace Safety and Security*, 2012.

[15] H. Gao, J. Hu, C. Wilson, Z. Li, Y. Chen, and B. Y. Zhao, "Detecting and characterizing social spam campaigns," in *ACM IMC*, 2010.

[16] X. Qi, L. Nie, and B. D. Davison, "Measuring similarity to detect qualified links," in *ACM Workshop on Adversarial information retrieval on the web*, 2007.

[17] M.-Y. Kan and H. O. N. Thi, "Fast webpage classification using url features," in *ACM CIKM*, 2005.

[18] E. Baykan, M. Henzinger, L. Marian, and I. Weber, "Purely url-based topic classification," in *ACM WWW*, 2009, pp. 1109–1110.

[19] A. A. Goshtasby, *Similarity and dissimilarity measures*. Springer, 2012.

[20] W. W. Cohen, P. Ravikumar, and S. E. Fienberg, "A comparison of string distance metrics for name-matching tasks," in *IJCAI-03 Workshop on Information Integration*, 2003, pp. 73–78.

[21] V. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals." in *Soviet physics doklady*, 1966, pp. 10–707.

[22] A. Finamore, M. Mellia, M. Meo, M. Munafo, and D. Rossi, "Experiences of Internet Traffic Monitoring with Tstat," *IEEE Network*, vol. 25, pp. 8–14, May 2011.

[23] I. Trestian, S. Ranjan, A. Kuzmanovic, and A. Nucci, "Googling the internet: Profiling internet endpoints via the world wide web," *IEEE/ACM Trans. Netw.*, vol. 18, no. 2, pp. 666–679, Apr. 2010.

[24] H. Metwalley, S. Traverso, and M. Mellia, "Using Passive Measurements to Demystify Online Trackers," *IEEE Computer "Communications and Privacy under Surveillance Issue"*, vol. 49, pp. 50–55, 2016.