Clustering and evolutionary approach for longitudinal web traffic analysis

Clustering and Evolutionary Approach for Longitudinal Web Traffic Analysis

Andrea Morichetta, Marco Mellia

Politecnico di Torino
andrea.morichetta@polito.it, marco.mellia@polito.it

Abstract

In recent years, data-driven approaches have attracted the interest of the research community. Considering network monitoring, unsupervised machine learning solutions such as clustering are particularly appealing to let the network analysts observe patterns, and track the evolution of traffic over time. In this paper, we present a novel unsupervised methodology to automatically process and analyze batches of HTTP traffic, looking just at the URL structure. First, we describe *IDBSCAN*, Iterative-DBSCAN. We design it to obtain well-shaped clusters, and to simplify the choice of parameters – often a cumbersome step for the network analyst. Second, we show LENTA, Longitudinal Exploration for Network Traffic Analysis, which allows to automatically observe the evolution over time of traffic, naturally highlighting trends and pinpointing anomalies.

We first evaluate IDBSCAN and LENTA on synthetic data to compare their performance against well-known algorithms. Then we apply them on a real case, facing the analysis of hundred thousands of URLs collected from a live network. Results show both the goodness of clusters produced by IDBSCAN and LENTA ability to highlight changes in traffic, facilitating the analyst job.

Keywords: Big data, Clustering, Edit Distance, Machine Learning, Security, Traffic Monitoring

1. Introduction

Unsupervised machine learning techniques, and in particular clustering, are popular approaches to automatically mine data and discover eventual patterns by grouping similar elements together [1]. Density-based clustering techniques offer the possibility to automatically find dense areas, *i.e.*, regions in which there is a number of objects with similar characteristics, where clusters of any shapes naturally emerge. DBSCAN [12] is the most known implementation and it is widely used in different applications, including network traffic analysis [11, 23]. In contrast to supervised approaches, clustering does not require that points are already assigned a label, *i.e.*, a class. This makes clustering appealing for

network traffic and web monitoring, where the heterogeneity of applications, terminals, and user interests makes it difficult to obtain labeled datasets.

In recent years, we witnessed the consolidation of internet services toward the usage of HTTP at the application layers, making this protocol the de-facto new "narrow waist" of the internet [29]. Video streaming, music, VoIP, chat, and traditional web browsing today run on the top of HTTP or HTTPS. Even malware prefers HTTP as a protocol to let infected clients communicate to command and control (C&C) servers [3]. This originates from the easiness for HTTP traffic to bypass network firewalls and intrusion prevention systems. While this has simplified the structure of the protocol stack, the complexity of modern services has complicated the analysis of web traffic, so that it is very hard to understand how services are running in the network. In this paper, we focus on web traffic, and specifically on the analysis of URLs that browsers uses when fetching objects from the web.

Clustering algorithms, in this context, allow one to reduce the size of the problem from a hundred thousand single objects – the unique URLs – to few hundreds of clusters, which contain "similar" URLs. Notice that most URLs carried by a network are not generated by an intentional user action (*e.g.*, the click of a link on a page), but are instead due to browsers and applications fetching objects in a web page, or system component for a web-app [34], including malwares that periodically contact C&C server or execute automatic actions. These URLs have often a regular syntax, which makes them strictly different, but similar in the format. Designing a clustering solution for URLs requires ingenuity, given URLs are strings, for which the notion of similarity and distance is not trivial to define.

Furthermore, off-the-shelf solutions like DBSCAN require the user to set parameters which are hard to choose. In particular, the choice of the radius to consider an area as dense is often cumbersome to make, especially if the final user does not have a clear knowledge on the feature space and of the distances used. Clustering results thus inaccurate, compromising the final outcome.

In this paper we describe and test a novel clustering algorithms, Iterative DBSCAN - IDBSCAN, that we previously introduced in [27]. It builds on DBSCAN and controls the quality of the clustering by automatically selecting parameters and iterating over data multiple times. The results is a clustering algorithm that produces very homogeneous clusters while being very robust to parameter settings. We demonstrate this using both benchmarking datasets and a use case entailing HTTP URLs collected from an operational network and comparing it to other clustering solutions.

We next design a system that is able to track traffic evolution over time, automatically identifying traffic produced by already known applications, and pinpointing new traffic. We introduced it in [27], with the name of LENTA (Longitudinal Exploration for Network Traffic Analysis). It builds on IDBSCAN, and a novel self-learning approach that lets the system build and update its knowledge. This knowledge grows thanks to a comparison methodology, which associates clusters obtained from a new snapshot of data with previously observed clusters. In this way, LENTA offers the analyst only new and previously

undetected clusters, while traffic in previously known clusters is automatically labeled. This system highlights changes and pinpoints the birth of previously unseen traffic patterns, building a longitudinal view of traffic.

LENTA results in a comprehensive solution that lets the network analyst automatically process batches of URLs and discovers similarities, new applications, and anomalies, including malware or malicious traffic.

We test LENTA on HTTP traffic collected by a passive probe that observes thousands of users in an ISP network during one week. Our prototype is able to process one day worth of traffic in slightly more than two hours. Results show both IDBSCAN ability in creating few and homogeneous clusters, which are easy to investigate and associate to services or malicious activities, and the capability of LENTA to identify new traffic generated by previously unknown systems. For instance, in our experiment LENTA discovers traffic related to well-known services (*e.g.*, CDN, video services, online tracking and advertisement systems), unexpected applications for the considered environment (*e.g.*, Chinese chatting applications) and even traffic generated by infected machines (*e.g.*, malware contacting C&C servers).

These results show the potential of LENTA to support the analysis and discovery of services running on the top of HTTP, and to help the security analyst in understanding current web services.

In our previous work we introduced CLUE [26], an unsupervised technique for mining URLs to let the analyst identify possibly malicious traffic or other services, which leverages DBSCAN. In [27] we improved the clustering algorithm and introduced LENTA to identify variations in time and help the analyst in finding suspicious traffic that may require further investigations by the network analyst. In this work, we extend our previous works by (i) providing a thorough parameter settings of IDBSCAN, (ii) adding a comparison with popular clustering algorithms, (iii) tuning LENTA using both synthetic and real case datasets. Thanks to this extensive analysis we prove the robustness and flexibility of our approach, comparing their performances with respect to other techniques. The paper is organized as follows: Sec. 2 provides an overview of the scenario in which we operate detailing the kind of data and introducing the system with an overview. Sec. 3 provides a description of the intuitions behind the design of LENTA. Sec. 4 and Sec. 5 give a detail of the performance of IDBSCAN and LENTA over different data set and in comparison with different techniques. Sec. 6 details the results of IDBSCAN on one day of traffic, while Sec. 7 describes the application of LENTA over one week of analysis. Sec. 8 presents the related works, before drawing conclusions in Sec. 9.

2. Motivation and System overview

In this paper, we describe our system LENTA and the motivations behind our technical choices, detailing each block and analysing its performance, providing benchmarks with respect to other approaches.

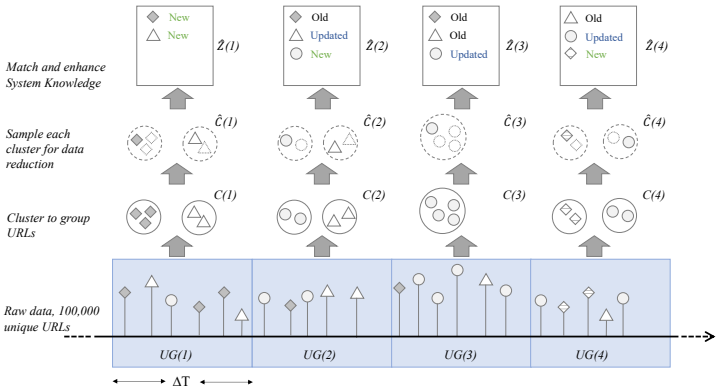


Figure 1: System overview

2.1. Motivation

The analysis of HTTP traffic, which still today amounts to more than 40% of web traffic according to global statistics [19], requires specific tools and methodologies. In this work, we leverage string similarity to generate homogeneous groups of URLs instead of simply merge together those elements that have, *e.g.*, a common domain name. Ideally, we aim at grouping together all those URLs that refer to the same service, *e.g.*, videos from Netflix, or services offered by the same cloud platform, or requests to a botnet C& C server.

To give the reader the intuition of our problem, Table 1 shows examples of URLs. *A1*, *A2* and *A3* belong to the same malware called TidServ – identified in our dataset using a professional IDS. All URLs have common substrings in the object path, but strictly different domain names and URLs. This is a common behaviour in malicious applications which apply approaches to rapidly change the domain name to evade static blacklist-based controls, the so-called DGA (Domain Generation Algorithm) technique [5, 31]. *B1* and *B2* illustrate two URLs generated by Sony connected Smart-TVs which access the same service, but with different URLs. This is typical of services that use the same web platform and that can be interesting to point out. In both the above examples, we would like the algorithm to identify these regular patterns, and form two groups, one for the malware, one for Smart-TV traffic.

Notice that grouping by domain name is not sufficient. Indeed, some services are hosted on the same domain name, but are logically very different. This is the case of the third example, *C1* and *C2*, where *Google Flights* and *Gmail* URLs are shown. In this case, we would like to identify two groups, one for each service.

In this paper we propose LENTA to support network analysts in automatically observing those services. We leverage unsupervised machine learning, *i.e.*, clustering, based on the analysis of URL string. For this phase, we introduce

Table 1: Examples of similar URLs

Example of URL	Example class
swltcho81.com/[...]**VyPTQuMCZiaWQ9[...]**	A1
rammyjuke.com/[...]**VyPTQuMCZiaWQ9[...]**	A2
iau71nag001.com/[...]**VyPTQuMiZiaWQ9[...]**	A3
bravia.dl.playstation.net/bravia/WidgetBundles/BgmSearch-2ndDisp/info.xml	B1
applicast.ga.sony.net/WidgetBundles/SNY_RSSReader/icon.png	B2
google.com/flights/#search;f=TRN,ITT,TPY;t=LAX;d=2018-01-22 ;r=2018-01-26	C1
google.com/mail/u/0/#inbox/160c745d9e5f6684	C2

IDBSCAN, an iterative clustering technique in which the quality of the clusters is guaranteed and clustering is reapplied to those objects whose grouping is poor.

The excellent clustering results are our building block to design an evolutionary methodology that can automatically highlight changes in traffic over time, looking at those patterns that are formed, which evolve and disappear, so to provide a general framework that automatically adapts to traffic.

2.2. System Overview

Fig.1 sketches the overall process of LENTA. It ets as input all URLs extracted by a network during a predefined period of time. URLs may be extracted from passive probes that sniff packets, or by HTTP proxies, which log requests from clients. To get visibility into HTTPS traffic, men-in-the-middle proxies could be used.

Our goal is to group all those URLs in clusters by only looking at the URLs themselves. For this, we process URLs in batches, $UG(i)$, where we insert all unique URLs seen during the *i-th* time interval of a desired amount of time ΔT. Only unique URLs are considered since our goal is to understand which resources are fetched by clients, independently of their popularity. At the end of a period, URLs form the clusters $\mathcal{C}(i)$. Several challenges arise here, from the computation of the similarity between two URLs (*i.e.*, strings), to the proper choice of the clustering algorithm, from the parameter settings to a scalable design.

Once clusters are identified, we reduce the number of URLs by applying a sampling process, *i.e.*, by extracting a summary of URLs found in each cluster, obtaining in output $\hat{\mathcal{C}}(i)$. This has the benefit to reduce the footprint of the data and limit the computational complexity of the next steps.

Next, we compare clusters found in the current batch with those found in the past, $\hat{\mathcal{Z}}(i-1)$, which are stored in the System Knowledge. If no match is found, then the current cluster is considered new and automatically added to

the System Knowledge after being inspected by the network analyst, that can provide a meaningful label. As we will show, the labeling process is greatly simplified by the availability of several URLs of the same type that let a domain expert take informed decisions.

3. Methodology

Here we detail our methodology defining the techniques adopted at each step. We start describing IDBSCAN. We then illustrate the sampling technique. Next, we detail operations correlated to the update and support of the System Knowledge. At last, we discuss proper definition of distances to compute similarity among URLs.

3.1. Clustering

In the field of unsupervised methods, clustering is one of the most popular. This class of algorithms aims at grouping together similar elements of a dataset, separating them from the others. The three main categories of clustering are partitional, hierarchical and density-based. Describing all of them is beyond the scope of this work. However, it is important to highlight the main features that influenced the decision of picking the density-based category as our choice. First of all, density-based clustering algorithms work with different types of distance metrics, included not Euclidean ones. Secondly, they do not require to specify a priori the desired number of clusters. Lastly, they allow the presence of noisy points, *i.e.*, points that are not assigned to clusters.

3.1.1. DBSCAN

For clustering, we built upon and improve the well-known DBSCAN algorithm [1]. A cluster is identified as the concatenation of consecutive dense areas in the data space. Given an object o, its density can be measured by the number of elements close to it. DBSCAN finds the core points, that are those objects that have dense neighborhoods; then it connects these core points and their neighbors to form the dense regions, *i.e.*, the clusters. To define the neighborhood area, the ϵ parameter is used. This represents the radius of the sphere that has o as the center. A neighborhood is dense if there are at least MinPoints in the sphere of radius ϵ.

3.1.2. Iterative DBSCAN (IDBSCAN)

Despite the good results, the setting of the MinPoints and ϵ parameters remains difficult. In particular, MinPoints can be reasonably set using domain knowledge since it represents the minimum number of elements to form a cluster. ϵ is instead hard to set, especially if the used distance is not well understood. In the original CLUE [26], ϵ was manually selected, a cumbersome and error-prone task. Here we propose a new approach to automatically compute ϵ, while also improving the final clustering. The intuition is to iteratively run DBSCAN, each time using a different setting of ϵ, and each time accepting only those clusters

that, after an evaluation with a quality measure index, result to be well-shaped. Objects in bad-shaped clusters are eventually re-clustered in the next iteration, with a different choice of ϵ. This produces a remarkable improvement of LENTA's clustering stage, by further splitting/merging clusters at each iteration, until they eventually form well-shaped clusters. After a maximum number of iterations, or in case of a dead loop, the algorithm stops and labels all the remaining elements as noise points (*i.e.*, not assigning them to any cluster). Those are outliers that would have to be ignored.

We define ϵ by using an a-priori rule, *i.e.*, we want the algorithm to cluster a given percentage η of objects at each iteration. To choose the proper ϵ that would guarantee this, we rely on the k-Distance graph rule [1]. Let $k = \text{MinPoints}$. For each object $i = 1, \ldots, N$ in the current dataset, the k-th nearest point is found, whose distance is d_i. We next sort $\{d_i\}$ from the lowest to the highest distance and look for the minimum threshold d_{th} for which $d_i < d_{th}$ for a certain fraction of points η. We set $\epsilon = d_{th}$. With this choice, a η percentage of objects have at least $k = \text{MinPoints}$ objects at a distance smaller than ϵ. Those would become core points, and form a cluster.

To identify well-shaped clusters, we rely on the *silhouette analysis*, an unsupervised cluster evaluation methodology to find how well each object lies within its cluster [30]. The silhouette coefficient $s(i)$ measures how close the point $i \in C$ is to other points in C, and how far it is from points in other clusters. Let $a(i)$ be the average distance of point i with all points in its cluster. Let $b(i)$ be the minimum among average distance of point i to points in other clusters. In formulas, we have:

$$a(i) = \frac{1}{\|C\|} \sum_{j \in C \neq i} d(i,j) \tag{1}$$

$$b(i) = \min_{C' \neq C} \left(\frac{1}{\|C'\|} \sum_{j \in C'} d(i,j) \right) \tag{2}$$

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{3}$$

It results in $s(i) \in [-1, 1]$. Values close to 1 indicate that the sample is far away from the other clusters, and very close to all other 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 clusters. Finally, negative values of $S(C)$ indicate that i might have been assigned to the wrong cluster. The average $S(C) = E[s(i), i \in C]$ over all the points in cluster C is a measure of how tightly grouped all the elements in C are.

Given a cluster C, we say it is well-shaped if $S(C) > S_{min}$. S_{min} represents the minimum value for the cluster silhouette. This parameter provides a threshold that allows separating good clusters from those that require further processing. Consequently, if C is well-shaped, we insert C in the set of clusters found so far. Otherwise, we put all points in C in the remaining set of points

to be considered for the next iteration of clustering. By setting a maximum number of iterations, we avoid dead loops. If there is a cluster that cannot be rearranged, the remaining elements are labeled as noise.

At the end of iterations, we are guaranteed to have all well-shaped clusters, with the final clustering $\mathcal{C}$ being

$$\mathcal{C} = \bigcup_j \{C_j | S(C_j) > S_{min}\} \tag{4}$$

3.2. Sampling for Data Reduction

Once clusters are formed, we sample a subset of elements from each of them. The rationale is twofold: to ease the comparison between clusters by reducing computational complexity while maintaining their information quality; and to keep in the System Knowledge a digest of the collected traffic, thus reducing its footprint. Indeed, each time we need to look for similar clusters in our System Knowledge, we need to compute the distance between all the points of the cluster that we are considering, and all the other points in the clusters stored in the System Knowledge. To reduce the number of points (and thus complexity), we rely on sampling to limit the number of elements in the System Knowledge.

Notice that both the clustering and the System Knowledge look up must be completed in ΔT, *i.e.*, before the next batch of data arrives. For this, it is important to guarantee that the samples are representative of the clusters' content. With this goal, we propose and test three different approaches. We sample each cluster $C_j \in \mathcal{C}$ keeping either a ratio $r \in [0, 1]$ of the cluster population, or a fixed specimen. At the end of the process, a set of sampled clusters $\hat{\mathcal{C}} = \bigcup_j \hat{C}_j$ is obtained.

Let m be the number of elements to extract. In case of fixed ratio r, we set $m = \lceil r||C_j|| \rceil$, and then pick $\hat{C}_j = sample(C_j, m)$. In case of a fixed sampling, we choose m a priori, and we select elements as $\hat{C}_j = sample(C_j, m)$.[1]

$sample(C_j, m)$ is a function that extracts m samples from C_j. We consider two samplings:

- Random sampling: selecting m objects at random from the elements of C_j, *i.e.*, $sample(C_j, m) = rand(C_j, m)$;
- Percentile sampling: selecting the elements that best represents the different kind of objects present in a cluster, *i.e.*, $sample(C_j, m) = percentile(C_j, m)$.

$percentile(C_j, m)$ extracts m representatives by looking at the distribution of mean distances for each object $s_i \in C_j$

$$\left\{ E_{s_k \in C_j}[d(s_i, s_k)], \forall s_i \in C_j \right\} \tag{5}$$

The selected elements correspond to values that divide in equally sized sets the cluster, *i.e.*, that correspond to the m percentiles. The idea behind percentile selection is to have a set of cluster's subsamples that includes both elements

[1] In case $|C_j| \leq m$, all elements are selected.

that are in the center area of a cluster and the ones at its border. Note that in case of $m = 1$, $percentile(C_j, m)$ would select the so-called medoid, *i.e.*, the element whose average dissimilarity to all the objects in the cluster is minimal.[2] The medoid is generally a good choice to describe a group of elements, but it is more appropriate for spherical and homogeneous clusters. Since a cluster in IDBSCAN is made by a chain of interconnected smaller spherical dense areas, the choice of only one point would exclude other possibly interesting instances. In this sense, the percentile sampling produces a sampling that better represents the population of the cluster.

3.3. System Knowledge enhancement intuition

LENTA maintains the set of clusters found in the past in the System Knowledge $\hat{\mathcal{Z}}(t)$, t being the time slot. At the beginning $\hat{\mathcal{Z}}(0) = \emptyset$. Given a sampled cluster $\hat{C}_i$ we want to identify the closest cluster found in the past. Let

$$d_{min}(\hat{C}, \hat{\mathcal{Z}}) = \min_{\hat{Z} \in \hat{\mathcal{Z}}} \left(d\left(\hat{C}, \hat{Z}\right)\right)$$

$$\text{where } d(\hat{C}, \hat{Z}) = \min_{\substack{c \in \hat{C} \\ z \in \hat{Z}}} d(c, z) \quad (6)$$

Equation 6 explains how to find the closest cluster for $\hat{C}_i$. We can call it $\hat{Z}_l$. $\hat{Z}_l$ is the cluster belonging to the System Knowledge $\hat{\mathcal{Z}}(t)$ that has the lowest value of $d\left(\hat{C}, \hat{Z}\right)$, *i.e.*, is at the minimum distance from $\hat{C}_i$. $d\left(\hat{C}, \hat{Z}\right)$ is extracted computing all the possible pairs of distances between the elements of $\hat{C}_i$ and the elements of $\hat{\mathcal{Z}}(t)$.

Let $\hat{\mathcal{C}}(t)$ be the result of the clustering of the current batch. We need to check if a cluster $\hat{C}_j(t) \in \hat{\mathcal{C}}(t)$ has been already found in the past, or if it represents new traffic. For the cluster $\hat{C}_j(t)$, the most similar cluster $\hat{Z}_l(t-1) \in \hat{\mathcal{Z}}(t-1)$ is

$$\hat{Z}_l(t-1) = \arg\min \left(d_{min}\left(\hat{C}_j(t), \hat{\mathcal{Z}}(t-1)\right)\right) \quad (7)$$

A cluster is then considered as new if the minimum distance is larger than the threshold α. The System Knowledge is updated as follows:

$$\hat{\mathcal{Z}}(t) = \hat{\mathcal{Z}}(t-1) \cup \left\{ \hat{C}_j(t) \in \mathcal{C}(t) \mid d_{min}\left(\hat{C}_j(t), \hat{\mathcal{Z}}(t-1)\right) \geq \alpha \right\} \quad (8)$$

That is, we add a new cluster found at time t if its distance to the closest cluster is higher than α.

3.4. Ageing

When $d_{min}(\hat{C}_j(t), \mathcal{Z}(t-1)) < \alpha$, two clusters are considered similar, so they contain the same kind of information. The new cluster is associated to the old one and may contain new knowledge, *e.g.*, some important changes in the

[2] The medoid is different from the centroid since the first is selected among the elements of the cluster.

particular service or differences in the structure or information carried by the considered objects, that in our use case are URLs. It is vital to register, if possible, those updates.

We use a random replacement policy. That is, we substitute each element $z_i \in \hat{Z}_l(t-1)$ with the element $c_i \in \hat{C}_j(t)$ with a certain probability p. So,

$$z_i := c_i \leftarrow p \;\; \forall i \in [1, m], \;\; z_i \in \hat{Z}_l(t-1), c_i \in \hat{C}_j(t) \tag{9}$$

In doing so, we update the System Knowledge clusters, ageing and replacing "old" representatives with fresher information.

3.5. Distance definition

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 [14]. The information that can be extracted by the distance computation can then be used for the clustering step.

For multi-dimensional points space the Euclidean Distance is the most well-accepted choice. It defines the dissimilarity between two objects as their actual geometric distance.

In the URL use 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 of 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.

Here, we focus on a particular class of distance metrics, the edit-distance based functions [10]. 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] $d_{LEV}(s_1, s_2)$ that assigns a unitary cost for all editing operations, *i.e.*insert, remove, or replace one character. It computes an absolute distance between 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.

The Levenshtein distance $d_{LEV}(|s_1|, |s_2|)$ is defined as:

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

where i and j are respectively the lengths of s_1 and s_2, *i.e.*$|s_1|$ and $|s_2|$, respectively, $d_{LEV}(i,j)$ is the distance between the first i characters of s_i and the

first j characters of s_2, and 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_{URL}. URLs are possibly very long strings with up to thousands of characters. For this we explicitly consider the string length, and normalize the results in a $[0, 1]$ range:

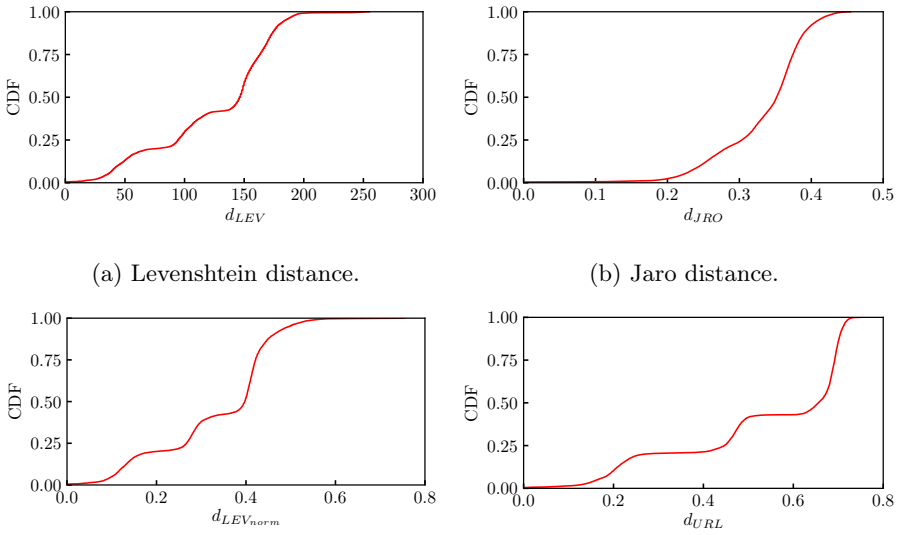
$$d_{URL}(s_1, s_2) = \frac{d_{LEV^*}(s_1, s_2)}{(|s_1| + |s_2|)}$$

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. We count the total number of insertions and deletions, but we weight replacement by a factor of two to count it as a deletion and insertion operation. We call it d_{LEV^*}.

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 $d_{LEV}(s_1, s_2) = 2$, 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 $d_{JRO} = 0.094$. $d_{LEV_{norm}}$ is the normalized version for the Levenshtein distance; as we analyzed above, d_{LEV} is 2, thus $d_{LEV_{norm}} = 0.095$. 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 that have been generated by TidServ, a polymorphic DGA malware. We then compute the distance between any pair of URLs (u_1, u_2). Fig. 2 shows the Cumulative Distribution Function (CDF) of the measured distances for d_{LEV}, d_{JRO}, $d_{LEV_{norm}}$ 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_{LEV} shows three groups in its CDF, suggesting for potential clusters. However, d_{LEV} 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-uniform shape, showing no clear steps that would



(a) Levenshtein distance.

(b) Jaro distance.

(c) Normalized Levenshtein distance.

(d) URL distance.

Figure 2: CDFs of distances on the TidServ URLs.
To facilitate the consequent clustering of URLs, we aim at having distances concentrated in ranges.

help in separating close from far away pairs.

$d_{LEV_{norm}}$ and d_{URL} satisfy the intuition of having distance ranges, as clearly shown by three modes in the CDF. Moreover, their support is bounded in the [0:1] range, normalizing the distance with respect to the length of the two considered strings. The results of the two approaches are similar, however we opt for d_{URL} since, as we can notice comparing Fig. 2c with Fig. 2d, the latter is able to produce larger values of distances, easing thus the separation of two elements. In fact, $d_{LEV_{norm}}$ range is smaller than d_{URL}. For example, considering the two TidServ elements *T1* and *T2*, where:

- **T1:** `lkeopee32.com/daz0yDNl785yiCU5dmVyPTQuMCZiaWQ9YjZjYWVhNjE0NjhhMmQ4ZTc0OGQ3ZTEzMTIyMDZiMDQ4NWY2MjJhYSZhaWQ9NDAxOTcmc2lkPTAmcmQ9MCZlbmc9d3d3Lmdvb2dsZS5pdCZxPXVuaWZlIG15ZGVzaw==07c`
- **T2:** `switcho81.com/JKm2Ptpd8J5xOYc1dmVyPTQuMCZiaWQ9YjZjYWVhNjE0NjhhMmQ4ZTc0OGQ3ZTEzMTIyMDZiMDQ4NWY2MjJhYSZhaWQ9NDAxOTcmc2lkPTAmcmQ9MCZlbmc9d3d3Lmdvb2dsZS5pdCZxPWFydWJhIG1haWw=37g`

The results that we obtain with $d_{LEV_{norm}}$ and d_{URL} are respectively: $d_{LEV_{norm}} = 0.11$, $d_{URL} = 0.19$.

4. IDBSCAN Benchmarking

4.1. Datasets

In this section we evaluate the performance of IDBSCAN using two datasets. First, we consider synthetic datasets commonly used for benchmarking. Then, we use a real dataset composed of URLs5.

For benchmarking and comparison with state of art algorithms, we generate data using the Python machine learning5 library *Scikit-Learn*.[3] We rely on the *make blobs* module, which creates globular clusters, generated by specifying the total number of points, the number of desired clusters, the feature space dimension, and the standard deviation of the points in each clusters. For our benchmarks, we consider datasets generated respectively (i) varying the number of clusters from 10 to 100 while keeping the dataset size equal to 20 000; or (ii) varying the size of the dataset keeping fixed number of clusters to 100 – maintaining the ratio between number of objects and number of clusters equal to 200. In both cases, we consider three-dimensional feature space, and standard deviation of points equal to 0.4.

As use case, we use actual URLs extracted from web traffic. In this work, we rely on Tstat [33], a scalable passive network monitoring solution that is able to process data in real time on high-speed links. Tstat implements an efficient DPI architecture that logs HTTP requests observed in packets. The logs contain details about observed HTTP transactions. For each transaction,it records the URL, a client identifier as well as other HTTP headers of requests and response, including the object path for not encrypted traffic. For our experiment we use a one-week-long HTTP trace collected during March 2016 in an ISP network. From this traces, we selected 30 households at random.[4] Only URLs are extracted, in aggregated form to make impossible to know which host of which household actually originated the request. K-anonymity - with k=30 - is guaranteed. To protect users privacy, all parameters in the URL have been removed, and only unique URLs were saved.[5] We observe more than 430 000 unique URLs during a week, more than 60 000 per day. For our preliminary benchmark we focus on the first day of traffic.

4.2. IDBSCAN parameter sensitivity

To understand the behavior of IDBSCAN and its sensitivity to parameters settings, we start to test performance over a synthetic set of 20 000 points, which form 100 clusters. We then vary the η and S_{min} parameters, keeping one fixed, respectively $\eta = 0.75$ and $S_{min} = 0.3$, and varying the other. Minpoints is set to 20 in all experiments. For performance indexes, we consider i) the number of clusters, ii) the homogeneity score, and iii) the Silhouette. Each configuration

[3] https://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets

[4] A home gateway connects all devices in a household to the Internet. The probe observes traffic generated by devices connected to the home network.

[5] The usage of this data set has been discussed and approved by our institution ethic committee, and by the ISP security group.

has been executed three times, each time with different random points. Plots report the mean value.

Fig. 3 reports the results. The plots on the left refer to $\eta \in [10, 90]\%$, $S_{min} = 0.3$. Results show that values of η higher than 60% generate very pure clusters, with homogeneity score close to 0.9, and Silhouette higher than 0.6. The number of clusters is in the 85-90 range (w.r.t. 100 ideal clusters). In a nutshell, IDBSCAN prefers very pure clusters, eventually discarding some few clusters which turn out to be not very pure.

Right plots report the sensitivity to S_{min} (with $\eta = 75\%$). Any value between 0.1 and 0.5 show negligible impact, with the performance that suddenly degrades when S_{min} is larger than 0.6. In a nutshell, IDBSCAN is very robust to S_{min}, so that any value in $[0.1, 0.5]$ range would be good.

For the goals of our work, we prefer to give importance to those settings that guarantee homogeneous and pure clusters, well separated to the others. For this, we fix $\eta = 0.75$ and $S_{min} = 0.3$.

4.3. Comparing IDBSCAN with other density-based technique

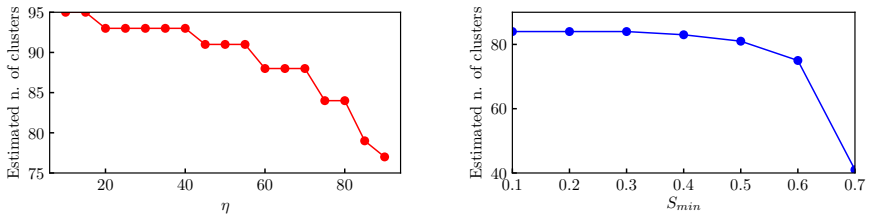
After verifying the potential of IDBSCAN for different parameters settings, we test it against other three popular density-based algorithms, namely DBSCAN [1], our baseline, OPTICS [4] and HDBSCAN [9, 25]. We briefly introduce them, referring the reader to the original publications for details.

OPTICS [4]. Ankerst et al., addressed the difficulty of setting DBSCAN parameters, and in particular the choice of ϵ. OPTICS stems from the basic idea that, given a fixed value for *MinPoints*, the clusters at higher density are contained in the ones that have lower densities. The idea is then to swipe ϵ to compute core points for high-density areas first, finding first the denser clusters. Points are thus ordered according to their density. Clusters can be identified graphically or automatically from that structure, and so different local densities may be defined to extract clusters in different areas of the data space.

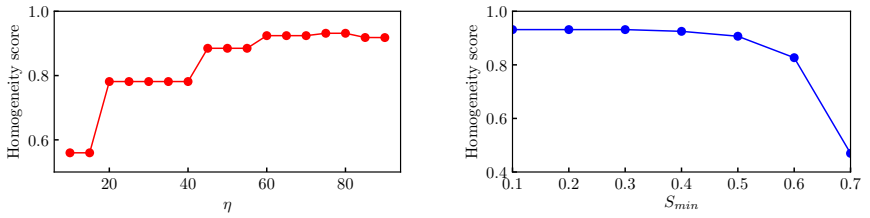
HDBSCAN [9, 25]. HDBSCAN aims at solving the ϵ setting in a similar way. It first creates a tree representation of all the possible clusters that can be generated for different ϵ. The algorithm then solves the problem of finding the best clusters as an optimization problem, where the overall stability of the clusters, defined following the Hartigan's definition of density-contour clusters [18], is maximized. Thanks to this approach, there is no need to tune the ϵ parameter.

Here we compare the performance of DBSCAN, OPTICS, HDBSCAN, and IDBSCAN over artificially generated datasets. We use the Scikit-Learn implementations of DBSCAN and HDBSCAN, OPTICS is executed using the pyclustering version, while IDBSCAN uses Scikit-Learn DBSCAN as building block.[6]

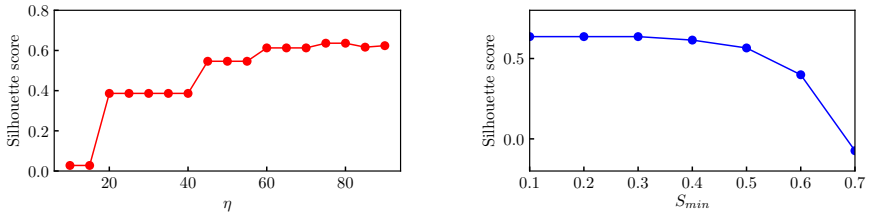
[6] `http://scikit-learn.org/`, `https://pypi.org/project/pyclustering`



(a) Number of clusters



(b) Homogeneity score



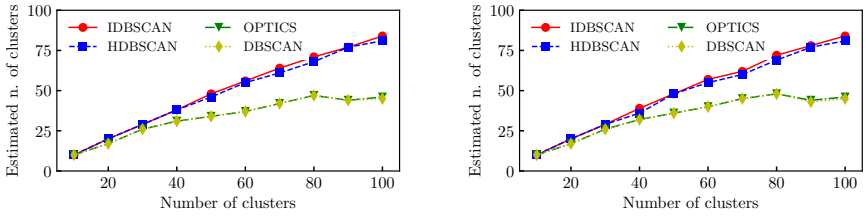
(c) Silhouette score

Figure 3: Evaluation of IDBSCAN over artificially generated datasets, varying η (left) and S_{min} (right).

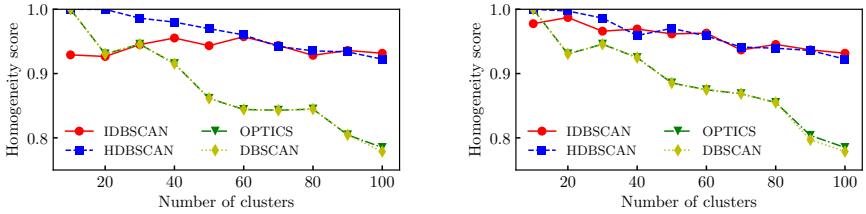
For all the algorithms we use MinPoints equal to 20 and, for DBSCAN and OPTICS, a ϵ value of 0.2.

Fig. 4 reports the results. We focus on two cases: On the left, we keep the total number of points fixed, and we split them over am increasing number of clusters; on the right, we keep the number of points per cluster fixed, and increase the number of clusters.

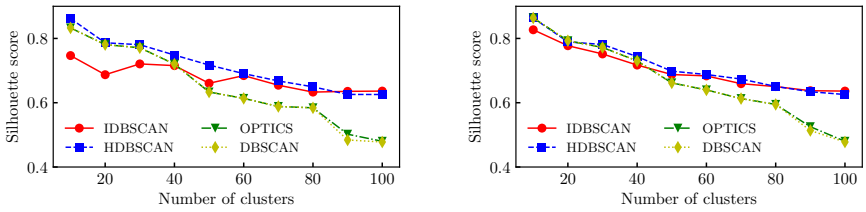
Results show that IDBSCAN and HDBSCAN outperform both DBSCAN and OPTICS, especially when the number of clusters is high. In those scenarios, HDBSCAN and IDBSCAN can identify a higher number of clusters (top plot), with the number of identified clusters that grows almost linearly with the actual number of clusters. Homogeneity and Silhouette (middle and bottom plots) are very good, too. In this benchmark, where data contains groups of points that



(a) Number of clusters



(b) Homogeneity score



(c) Silhouette score

Figure 4: Evaluation of density-based algorithms over artificially generated datasets. Plots on the left consider a fixed number of points, split over a varying number of clusters. Plots on the right refer to the case where the number of points per clusters is fixed.

are all of the same density, OPTICS performs identically to DBSCAN, given the pyclustering implementation that offers a threshold for ϵ. Not reported here, CPU time of OPTICS is also much higher than the one of HDBSCAN and IDBSCAN, with the original DBSCAN being the fastest, as expected.

4.4. Performance using URLs

We next compare IDBSCAN and HDBSCAN using actual data coming from our use case. We consider 60 000 unique URLs. We use MinPoints = 20 for both, $\eta = 75\%$ and $S_{min} = 0.3$ for IDBSCAN.

Here we do not have a ground-truth, and we do not know the correct number of clusters we shall identify. As such, we focus on the Silhouette as a metric to evaluate the goodness of the final clustering. Fig. 5 shows the results. It reports Silhouette for each cluster, ordered by increasing value. Overall, HDBSCAN identifies clusters (about 600), but 1/3 of those have a Silhouette value smaller than 0.3 – with about 100 clusters with negative Silhouette. We manually investigated those clusters, and clearly observed that those groups URLs do not fit well together. On the contrary, IDBSCAN returns fewer clusters (about 300), which are very pure – by construction with Silhouette higher then $S_{min} = 0.3$. In Sec. 6 we will show how informative these clusters are.

These results show how IDBSCAN outperforms HDBSCAN in our use case. It is able to identify few, but very pure clusters, simplifying the analyst job.

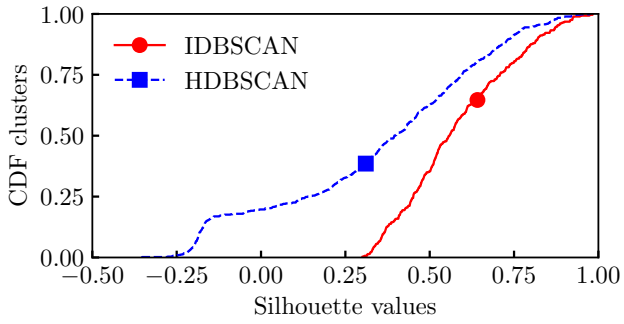


Figure 5: Silhouette of clusters obtained with IDBSCAN and HDBSCAN when considering one day of URLs.

5. LENTA Benchmarking

We now run a set of experiment to tune LENTA parameters. In particular, we focus on the sampling stage. Indeed, extracting a digest of the clusters via sampling represents the most critical step, since it is essential to balance representativeness while limiting the memory of the System Knowledge enhancement step.

We propose several methods for sample selection $sample(C_j, m)$: fixed size m, or proportional to the cluster dimension, with r ratio, and random or percentile sampling. We test those first using the synthetic dataset, and then with actual URLs.

5.1. Performance of sampling on synthetic data

We compare the sampling methodologies described in Sec. 3, over the results of the IDBSCAN clustering applied in the artificial dataset scenario, with 20 000 points, 100 clusters, and a three-dimensional feature space.

We set up an experiment where we split the dataset in two parts. The first part contains points of a subset of 50 clusters. The second dataset contains instead all points from all 100 clusters. We compute the clustering on the first set and extract the representatives. We then run again the clustering with all points from all 100 clusters, extract the representatives, and compare them with the previous representatives. The idea is to check if the clusters sampling technique allows the identification of the same clusters (the 50 clusters present in the first and second dataset), checking the similarity between the new representatives and the old ones. We repeat this experiment for the different sampling strategies, and for an increasing number of representatives.

Fig. 6 shows the results. For each test, we report, for each cluster C, the minimum distance $d_{min}(\hat{C}, \hat{\mathcal{Z}})$, sorted by increasing d_{min}. Ideally, we would expect to have $d_{min} = 0$ for those 50 clusters that are in common, while $d_{min} \geq \alpha$ for the 50 new clusters that are present in the second batch only.

Results show that the sampling and System Knowledge enhancement works quite well, with the percentile sampling performing better given its deterministic approach to select representatives. As we could expect, all of the three sampling methodologies perform better when increasing the number of representatives. The proportional approach seems to offer good results in this scenario. Recall that in this benchmark all clusters contain about 200 points each, so that also random sampling, and fixed sampling present good results.

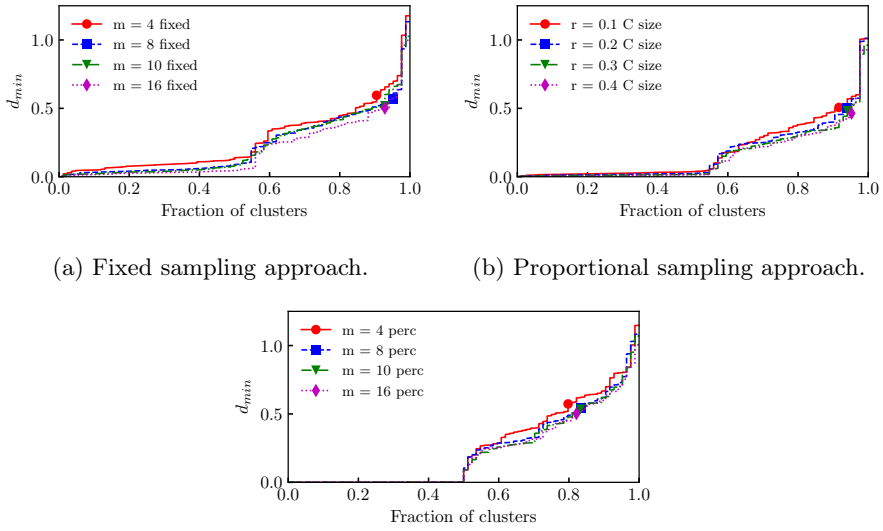
At last, considering the choice of the threshold α, we can see that any value higher than 0.1 would offer a good separation between the 50 old clusters, and the 50 new clusters.

5.2. Performance of sampling with URLs.

To choose which strategy works best in the real case scenario, we run a second experiment in which we again split the set of URLs in the first day of data into two sets. We run the clustering considering the first half of URLs, extract representatives for each identified cluster, and add them to the System Knowledge. We then rerun the clustering considering all URLs, extract representatives, and match the new clusters with those in the System Knowledge. In this case, too, we expect about 50% of clusters to be old, *i.e.*, $d_{min} < \alpha$, and 50% to be new, *i.e.*, $d_{min} \geq \alpha$.

Results are depicted in the plots of Fig. 7, which compares the three different sampling strategies, with different parameters. We clearly observe that the System Knowledge matching works as expected. The more the number of representatives, the more the ideal step-curve-behavior is visible. The approximation is very good, picking a fixed m equal to 16, and very similar to the step curve with proportional r of 20% or 30%.

Accordingly to both experiments, we obtain the best results when using percentiles, whose smart sampling guarantees optimal results. Indeed, when we consider the percentile, we always obtained a perfect distance of 0 for the clusters that contain the same elements of the compared ones. That is happening because two sets are equal and we deterministically select the representatives.



(a) Fixed sampling approach.

(b) Proportional sampling approach.

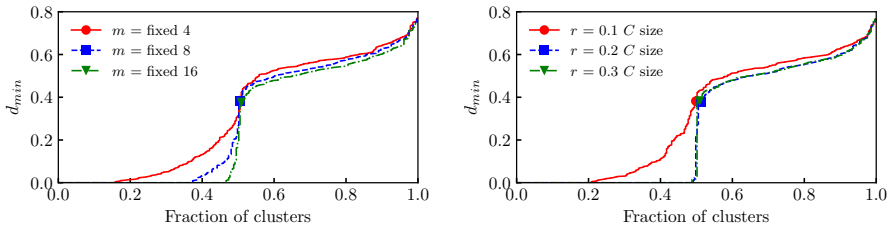
(c) Percentile sampling approach.

Figure 6: Sampling applied to the artificial dataset with 100 clusters, 50 of which were already seen in the past.

Considering the choice of α when actual URLs are considered, Fig. 7a, Fig. 7b and Fig. 7c clearly show that the new clusters tend to be very dissimilar from the old ones, and that any $\alpha \in [0.2, 0.4]$ is a proper choice. To not discard potentially new and interesting clusters, in the following we choose a value of $\alpha = 0.3$.

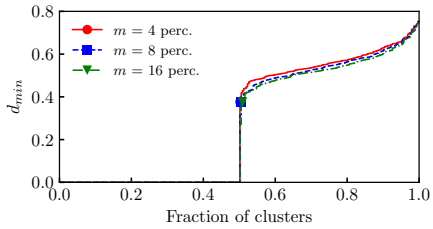
At last, enlarging the number of representatives has the drawback to increasing the computation complexity, due to the need to compute $O(m^2)$ $d_{URL}(.)$. Fig. 8 shows the experimental computational time using lin/log scales considering the set of 60 000 URLs. For variable fraction r, we choose the average number of elements in the cluster was chosen for a value of x. As expected, the curve grows quadratically for m (logarithmically in log scale), with $m = 32$ and $r = 20\%$ (on average 23 samples) or 30% (avg. 35 samples) that already have a complexity larger than 3 000 s. Without sampling, this experiments would require more than 7 hours to complete. Considering the System Knowledge would have thousands of clusters and that this step shall be completed every ΔT, the best trade-off between cluster similarity identification and computational time is obtained using a fixed $m = 16$.[7]

[7] In this case, too, CPU time can be reduced by computing d_{URL} in parallel.



(a) Fixed sampling approach.

(b) Proportional sampling approach.



(c) Percentile sampling approach.

Figure 7: d_{min} when 50% of traffic is the same and 50% is new. Different choices of sampling approaches.

6. Results

In this section, we provide experimental results for our use case. We choose $\Delta T = 24\,\text{h}$, $\eta = 75\%$, $S_{min} = 0.3$, $p = 0.2$ and $MinPoints = 20$ to look for well-shaped and big enough clusters. We tested different parameters, observing little changes.

Foremost we analyze the first day of traffic. LENTA obtains 283 clusters from the set of 59543 original unique URLs. The Silhouette coefficient $S(C)$ has a value of 0.5 or more for 183 clusters, with 55 of them with $S(C) > 0.75$. That is, clusters are very well shaped.

The top part of Tab. 2 shows the biggest clusters, while the bottom part those with the highest silhouette. The table reports the silhouette $S(C)$, the most frequent hostname (in brackets the total number of hostnames), the number of unique URLs, and the type of the service. Although the majority of clusters are relatively small, some contain a considerable number of distinct URLs and of different hostnames. That behavior is not to be taken as granted, as often the complexity of URLs structure tend to increment the distance also for actually similar elements.

Some suspicious clusters are also identified. For instance, 30 unique URLs form a cluster where URLs have all the same IP address `219.129.216.161` – but

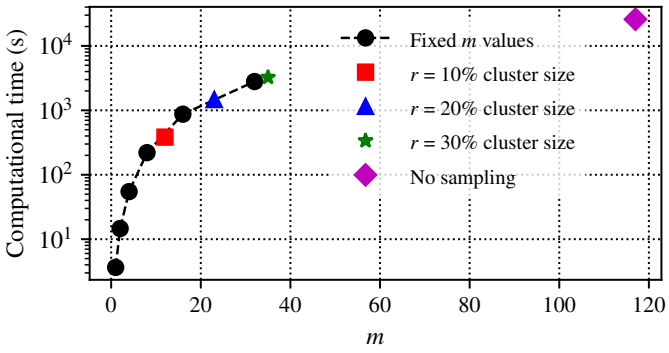


Figure 8: Computation time for different sampling strategies. Without sampling, the comparison of the System Knowledge would require too much time.

Table 2: Insight of the clustered HTTP traffic from the first day of analysis. On the top, the largest clusters. On the bottom, the top well-shaped clusters.

$S(C)$	**Main hostname (unique hostnames)**	**Elements**	**Activity**
0.52	scontent-mxp1-1.cdninstagram.com (4)	4359	Instagram CDN
0.92	se-rm3-18.se.live3.msf.ticdn.it (6)	3504	Entertainment - Streaming CDN
0.36	skyianywhere2-i.akamaihd.net (9)	2087	Entertainment - Streaming CDN
0.30	www.google-analytics.com (29)	1940	Tracking
0.95	rtinfinityh2-a.akamaihd.net:80 (1)	1227	Entertainment - Streaming CDN
0.76	videoassets.pornototale.com (1)	751	Adult content
0.57	tracking.autoscout24.com (2)	592	Tracking
0.37	ec2.images-amazon.com (10)	575	Image CDN
0.56	thumbs-wbz-cdn.alljapanesepass.com (1)	393	Adult Content
0.66	video-edge-8fd1c8.cdg01.hls.ttvnw.net (4)	359	Entertainment - Streaming
0.98	iframe.ad (1)	27	Advertising
0.97	news.biella.it (1)	23	News
0.95	rtinfinityh2-a.akamaihd.net:80 (1)	1227	Entertainment - Video Streaming CDN
0.93	motoitalia01.wt-eu02.net (1)	45	Tracking
0.92	skygo.sky.it (1)	45	Entertainment - Video Streaming
0.92	se-rm3-18.se.live3.msf.ticdn.it.msf.ticdn.it (6)	3504	Entertainment - Video Streaming CDN
0.92	219.129.216.161 (1)	30	Malware
0.92	a.applovin.com (1)	20	Analytics
0.92	rum-dytrc.gazzetta.it (1)	47	Entertainment - Analytics

apparently random paths. After further analysis[8], this cluster is actually found malicious. Other suspicious clusters emerge as well. At last, it is important to mention that the same service, *i.e.*, the same hostname, may be broken apart in multiple clusters, each one containing specific content. For example, the Chinese messaging system `msg.71.am` is divided into two clusters, one serving images

[8]Google results: https://goo.gl/q3DgT8, VirusTotal results https://goo.gl/fqrNkG

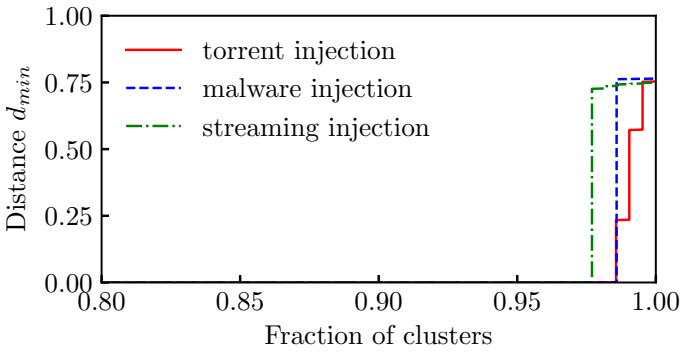


Figure 9: Curves of distances when new traffic is injected in the controlled experiment. Top 20% clusters are reported.

(.GIF), and the other exchanging control information like devices reports.

These results clearly show that LENTA let the services that commonly characterize the traffic emerge. The security analyst can then analyze clusters and consequently label them.

7. Evolution over time

In this section, we show the results of running LENTA in a real scenario. We first consider a controlled experiment and then we apply LENTA over 21 days of traffic collected from the ISP network.

7.1. In vitro experiment

To evaluate the reaction of LENTA with respect to the appearance of anomalous elements, we design a controlled experiment in multiple stages. We start from an initial group $UG(0)$ of almost 33 000 unique URLs extracted at random from the previous dataset. We then artificially create new groups $UG(1), UG(2)$ and $UG(3)$ where we progressively inject URLs belonging to different applications. We first add a block of 200 torrent URLs, *i.e.*, $UG_1 = UG_0 \cup \{TorrentURLs\}$. Next, we add 228 malicious URLs generated by hosts infected by *TidServ*, *i.e.*, $UG(2) = UG(1) \cup \{TidservURLs\}$. Finally, we inject 549 URLs generated by a popular streaming service, *i.e.*, $UG(3) = UG(2) \cup \{StreamingURLs\}$.

After each stage, we run LENTA and check if it is able to identify the new traffic. Results are reported in Fig. 9, which shows the minimum distance $d_{min}(\hat{C}(t), \mathcal{Z}(t-1))$ between clusters found in $UG(t)$ and those in the System Knowledge build on previous steps. We report only the first 20% of clusters, ordered by distance. As clearly shown, LENTA is able to recognize the new traffic: first, d_{min} is equal to zero for those clusters in $UG(t)$ that were already

Table 3: New clusters after the comparison with the System Knowledge.

Experiment stage	$\mathbf{d_{min}}$	**Main hostname(s)**
UG_1 Torrent	0.75	b-0.ad.bench.utorrent.com
	0.57	scorecardresearch.com, pixel.quantserve.com
	0.23	torrent.gresille.org
UG_2 Malware	0.76	wuptywcj.cn
	0.76	*-6nbcv.com, iau71nag001.com
	0.76	bangl24nj14.com, switcho81.com
UG_3 Streaming	0.75	198.38.116.148
	0.74	23.246.50.136, 198.38.116.148
	0.74	198.38.116.148
	0.73	23.246.50.136, 198.38.116.148
	0.72	198.38.116.148

Table 4: Behavior of the system during the week.

	Mar-01	Mar-02	Mar-03	Mar-04	Mar-05	Mar-06	Mar-07
Unique URL	59543	62842	67789	61849	77770	87928	88396
Daily Clusters	283	322	348	304	396	428	431
System knowledge	283	475	643	765	927	1097	1267
System enhancement	283	192	168	122	162	170	170

Table 5: Most interesting clusters obtained by the daily comparison with the system knowledge in the controlled experiment.

Day	**Main hostname (unique hostnames)**	**Activity**	**Day**	**Main hostname (unique hostnames)**	**Activity**
Mar-02	adnxs.com (3)	Advertising	Mar-03	ams1.mobile.adnxs.com (1)	Advertising
	www.bing.com (1)	Search Engine		ads1-adnow.com (3)	Advertising
	amazon.it (3)	E-commerce		uk-ads.openx.net (1)	Advertising
	doubleverify.com (9)	Advertising		c.3g.163.com (1)	Chinese Website
	mp.weixin.qq.com (1)	Chinese Website		googleapis.com (1)	Cloud Storage
Mar-04	banzai-d.openx.net (1)	Advertising	Mar-05	engine.bitmedianetwork.com (1)	uTorrent Adv
	dt.adsafeprotected.com (1)	**Hijacker**		**62.210.188.202:8777** (1)	**Suspicious Port**
	gvt1.com (3)	**Hijacker**		**adaptv.advertising.com** (1)	**Suspicious Adv**
	windowsphone.com (1)	CDN Marketplace		pubnub.com (16)	Messaging
	ocsp.digicert.com (1)	Certificate Inspection		**irs01.com** (1)	**Suspicious Tracking**
Mar-06	23.246.50.130 (5)	Netflix Italy	Mar-07	aww.com.au (2)	News
	198.38.116.148 (3)	Netflix Germany		*.liverail.com (1)	Advertising
	23.246.50.136 (3)	Netflix Italy		**spaces.slimspots.com** (1)	**Adware Attack**
	23.246.51.136 (2)	Netflix Italy		googleusercontent.com (2)	Page Translation
	178.18.31.55:8081 (7)	**Suspicious Streaming**		**s8.algovid.com** (1)	**Malicious Adv**

present in $UG(t-1)$. Second, and more important, the new traffic is clustered in totally different clusters, whose d_{min} is much higher than $\alpha = 0.3$.

In details, Tab. 3 depicts the results of the experiment. First, all clusters contain only new URLs injected in each step of the process. Second, notice that LENTA identifies multiple new clusters for each stage. This is welcome, since each cluster corresponds to a semantically different service. For instance, for

the video streaming case, each cluster corresponds to videos served for different platforms (iOS, Android, and PC), and torrent clusters correspond to different swarms and trackers. Third, $d_{min} > 0.3$ for all clusters but one in the Torrent data, for which $d_{min} = 0.23$. This cluster would be associated to a previously seen cluster. The association is correct, and URLs have a very similar syntax to the one already found and related to a tracker service, `tntvillage`.

7.2. Real case scenario

We now run LENTA on one week of data collected in an ISP network. Table 4 details results. Figure 10 shows the growth of the system knowledge $||\hat{\mathcal{Z}}(t)||$ over time (blue bars), and the daily amount of clusters that are added during the enhancement phase (red bars). The table gives the actual figures. During each day, $280 \div 430$ clusters are identified, with the variability due to the daily activity of users. Some of those correspond to clusters already present in the System Knowledge (typically more than 50-70%), which grows over time of fewer than 170 clusters per day. In a nutshell, LENTA is able to decrease the amount of information the security analysts have to process by 3 orders of magnitudes so that they have to inspect about less than 200 clusters per day instead of managing several tens of thousands of unique URLs.

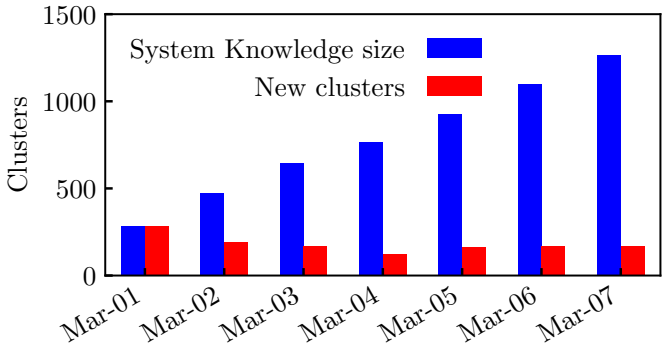


Figure 10: Daily enhancement of system knowledge

The variability of URLs grouped in the same cluster also simplifies the investigation of the service being involved. For instance, we checked some clusters that came into sight after each System Knowledge enhancement phase. We report, for each day, five new clusters among those that were reported to be among the most different with respect to the previously collected traffic, *i.e.*, those for which $d_{min}(C_j(t), \hat{\mathcal{Z}}(t-1))$ is higher.

Tab. 5 details the results. In this case, too, the services are related to streaming, advertising, e-commerce services. Some unexpected or at least not so frequent traffic emerges as well; for instance, on March 3rd, the `c.3g.163.com` cluster emerges. It is related to the Chinese web portal `www.163.com`, which was

never seen in the previous days. URLs are related to a newsfeed specific service. During March 4th and 5th, some suspicious or malicious traffic is identified. Clusters are related to hijacking services and aggressive advertisement targeting and are likely generated by some hosts infected by some malware. March 6th is extremely captivating. Eight out of ten most different clusters are formed by URLs characterized by IP addresses which resolve Netflix Italy or Netflix Germany CDNs. These were not found in the previous days, highlighting a change in the Netflix load balancing policies. The other cluster contains traffic from `178.18.31.55:8081`, connected to *liverepeater*, a keyword related to illegal streaming content. Finally, in the last day, some suspicious traffic is visible: uncommon services like `aww.com.au`, an Australian news website, and web pages translated using the Google Translate online service (curiously translating adult content website, possibly to evade content filtering policies).

8. Related Work

Clustering is considered one of the most interesting unsupervised learning techniques, providing a data structure partition which can be the basis for further learning. In particular unsupervised techniques have provided the possibility to obtain various forms of clusters and to separate the noisy points from the final result. However, the sensitivity to the parameters and the inability to manage data sets with different densities, has represented a limit. For these reasons other methodologies started to emerge. OPTICS [4] from Ankerst et al. offers a methodology to inspect the structure of clusters obtained from the data – the reachability plot – that is used to extract clusters looking at density valleys with visual inspection - a not always applicable solution. Campello et al. proposed HDBSCAN [9, 25] which is built on top of DBSCAN, and considers different radius values, combining the results to find the best clustering. However, it offers limited control on the quality of clusters. In our proposed solution IDBSCAN the clustering stage builds upon the classic DBSCAN and make use of silhouette to allow better results in terms of quality with respect to other well-known and novel algorithms. Furthermore, the automation of the choice of cumbersome parameters reduces the effort needed by the analyst to configure and tune the system.

Network traffic analysis, thank to the complexity and richness of its data, has witnessed in the last decade a lot of research concerning the use of machine learning techniques. The key applications are traffic classification and anomaly detection. In the latter case, the application of clustering technique is of great interest. Authors of [16, 17] addressed the task of botnets detection. The former uses a two-step clustering of communication flows, first coarsely grouping them considering a contraction of the feature space, and then, for each group, computing a more refined cluster where all the features are considered. The latter uses a hierarchical clustering technique to merge similar bags of bi-grams, extracted from messages collected from Internet Relay Chat monitoring, using the DICE coefficient to express the distance.

In other cases clustering is seen as a vehicle for classification. In [11] Erman et al. presented a work on the classification of network traffic, using transport layer statistics and testing clustering algorithms (namely, k-Means, DBSCAN and AutoClass) over different labeled datasets. Authors of [23] use labeled traffic flows and k-Means clustering for classification.

A number of studies focused on text and string mining techniques, with the goal of clustering network traffic. In the field of network security authors of [28] use a two-level clustering process, leveraging the single-linkage hierarchical algorithm to disclose similarities between malicious URL, with the goal of building malware signatures; Levenshtein distance, together with Jaccard Index, is used in the second clustering stage. In [20], semantic features of the URLs are used to target the same problem, using DBSCAN and Jaro-Wrinkler distance. Authors of [24] use the Levenshtein distance aiming at detecting phishing sites, whose names are built using typical spelling mistakes. Gao et al. [13] use clustering techniques to detect spam campaigns on Facebook, looking at similarities in destination URLs.

In this context, also big data approaches are starting to emerge to scale the analysis of traces [6, 7, 15, 22, 32]. They offer the ability to process massive data [22], and run machine learning methodologies for traffic classification [15, 32], traffic monitoring analytics [7], or in general to support the so-called data science process, *i.e.*, the extraction of insights from massive data [6]. Other works are exploring time series data to analyse user behavior, in different scenarios. For example, Bulut and Szymanski in [8] they perform an analysis on mobile user data trying to understand trends and deviation on users behaviors. In [2] Altman identify groups of Twitter subscribers looking at geographic location and similarities in the language used in tweets. This study identified features that contributes in the understanding of new spelling forms driven by the social network platform.

With respect to all these works, our solution LENTA aims to be a general purpose methodology that can be used in different fields and applications, with a scalable solution. Our goal is to examine the traffic to find, from URL structures, similar elements that can be used to easily identify patterns, anomalies and novelties in protocol, services and users behaviors.

Considering our previous research [26] [27], here we compared IDBSCAN, originally proposed in [27], with other clustering algorithms, both using synthetic datasets and real case scenarios. We extended thus the motivation behind our choices and we showed the potentials of our approach.

9. Conclusions and Future Work

In this paper, we presented LENTA, a methodology for the fast identification of HTTP-based service by looking at URLs string similarities. We designed a recursive version of a clustering algorithm over daily HTTP traffic generated by hosts in a network, called IDBSCAN. We tested it against other off-the-shelf algorithms, namely DBDSCAN, HDBSCAN, and OPTICS, using both a synthetic dataset and a real use case with URL objects, showing the benefits

of our proposed solution. We then performed the clustering algorithm for an entire week, comparing the result of each 24 hours with a collection of previously observed services. We found that LENTA allows reducing the traffic to manually check and to ease the observation of changes in the network behavior. Furthermore, it exposes well-formed clusters of URLs which greatly simplifies the identification of possibly malicious and undesired traffic.

This work goes in the direction of reducing the problem complexity, quickly producing an outcome for the analyst to whom are offered few hundreds of clusters instead of several hundred of thousands of URLs. Our results show that the methodology, applied in a long-term observation, is promising in the ability to identify anomalies in the traffic.

When considering HTTPS, LENTA would work with no changes, assumed visibility in HTTPS traffic is possible.In a corporate scenario this could be achieved using a MITM proxy, or directly instrumenting the browsers with a plug-in to log HTTP/HTTPS requests.

A supplementary effort is necessary to extend big data approaches to all the stage of the system to scale the analysis. Another possible follow-up work is the application of LENTA over different lexical features, like hostname in DNS queries, or user-agents in HTTP requests.

The system can be also applied on different scenarios, going from flow analysis to user-generated content characterization, *e.g.*, studying activity on social media platforms.

10. Acknowledgements

We want to thank the Vienna Science and Technology Fund (WWTF) through project ICT15-129, "BigDAMA" and the SmartData@PoliTO center for Big Data technologies that funded the research leading to these results.

[1] Aggarwal, C.C., Reddy, C.K.: Data Clustering: Algorithms and Applications. Chapman and Hall/CRC (2013)

[2] Altman, E.: Twitter and the Geo-linguistic fingerprint. (2012)

[3] Anderson, B.: Hiding in plain sight: Malware's use of tls and encryption. `https://blogs.cisco.com/security/malwares-use-of-tls-and-encryption`

[4] Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: Optics: ordering points to identify the clustering structure. In: ACM Sigmod record. Volume 28., ACM (1999) 49–60

[5] Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., Dagon, D.: From throw-away traffic to bots: Detecting the rise of dga-based malware. In: USENIX security symposium. Volume 12. (2012)

[6] Apiletti, D., Baralis, E., Cerquitelli, T., Garza, P., Giordano, D., Mellia, M., Venturini, L.: Selina: A self-learning insightful network analyzer. IEEE Transactions on Network and Service Management **13**(3) (Sept 2016) 696–710

[7] Baer, A., Finamore, A., Casas, P., Golab, L., Mellia, M.: Large-scale network traffic monitoring with dbstream, a system for rolling big data analysis. In: 2014 IEEE International Conference on Big Data (Big Data). (Oct 2014) 165–170

[8] Bulut, E., Szymanski, B.K.: Understanding user behavior via mobile data analysis. 2015 IEEE International Conference on Communication Workshop, ICCW 2015 (2015) 1563–1568

[9] Campello, R.J., Moulavi, D., Sander, J.: Density-based clustering based on hierarchical density estimates. In: Pacific-Asia conference on knowledge discovery and data mining, Springer (2013) 160–172

[10] Cohen, W.W., Ravikumar, P., Fienberg, S.E.: A comparison of string distance metrics for name-matching tasks. In: IJCAI-03 Workshop on Information Integration. (2003) 73–78

[11] Erman, J., Arlitt, M., Mahanti, A.: Traffic classification using clustering algorithms. In: Proceedings of the 2006 SIGCOMM workshop on Mining network data, ACM (2006) 281–286

[12] Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise.

[13] Gao, H., Hu, J., Wilson, C., Li, Z., Chen, Y., Zhao, B.Y.: Detecting and characterizing social spam campaigns. In: ACM IMC. (2010)

[14] Goshtasby, A.A.: Similarity and dissimilarity measures. Springer (2012)

[15] Grimaudo, L., Mellia, M., Baralis, E., Keralapura, R.: Select: Self-learning classifier for internet traffic. IEEE Transactions on Network and Service Management **11**(2) (June 2014) 144–157

[16] Gu, G., Perdisci, R., Zhang, J., Lee, W.: Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. (2008)

[17] Gu, G., Zhang, J., Lee, W.: Botsniffer: Detecting botnet command and control channels in network traffic. (2008)

[18] Hartigan, J.A.: Clustering Algorithms. Wiley, New York (1975)

[19] Khatouni, A.S., Trevisan, M., Regano, L., Viticchié, A.: Privacy issues of isps in the modern web. In: Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2017 8th IEEE Annual, IEEE (2017) 588–594

[20] Kheir, N., Blanc, G., Debar, H., Garcia-Alfaro, J., Yang, D.: Automated classification of c&c connections through malware url clustering. In: IFIP International Information Security Conference, Springer (2015) 252–266

[21] Levenshtein, V.: Binary codes capable of correcting deletions, insertions and reversals. In: Soviet physics doklady. (1966) 10–707

[22] Liu, J., Liu, F., Ansari, N.: Monitoring and analyzing big traffic data of a large-scale cellular network with hadoop. IEEE Network **28**(4) (July 2014) 32–39

[23] Liu, Y., Li, W., Li, Y.C.: Network traffic classification using k-means clustering. In: Computer and Computational Sciences, 2007. IMSCCS 2007. Second International Multi-Symposiums on, IEEE (2007) 360–365

[24] Maurer, M.E., Hofer, L.: Sophisticated phishers make more spelling mistakes: Using url similarity against phishing. In: Springer Cyberspace Safety and Security. (2012)

[25] McInnes, L., Healy, J.: Accelerated hierarchical density based clustering. In: 2017 IEEE International Conference on Data Mining Workshops (ICDMW). (Nov 2017) 33–42

[26] Morichetta, A., Bocchi, E., Metwalley, H., Mellia, M.: Clue: Clustering for mining web urls. In: 2016 28th International Teletraffic Congress (ITC 28). Volume 01. (Sept 2016) 286–294

[27] Morichetta, A., Mellia, M.: Lenta: Longitudinal exploration for network traffic analysis. In: 2018 30th International Teletraffic Congress (ITC 30). Volume 1., IEEE (2018) 176–184

[28] Perdisci, R., Lee, W., Feamster, N.: Behavioral clustering of http-based malware and signature generation using malicious network traces. In: NSDI. Volume 10. (2010) 14

[29] Popa, L., Ghodsi, A., Stoica, I.: Http as the narrow waist of the future internet. In: ACM Hotnets. (2010)

[30] Rousseeuw, P.J.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics **20** (1987) 53 – 65

[31] Schiavoni, S., Maggi, F., Cavallaro, L., Zanero, S.: Phoenix: Dga-based botnet tracking and intelligence. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer (2014) 192–211

[32] Suthaharan, S.: Big data classification: Problems and challenges in network intrusion prediction with machine learning. SIGMETRICS Perform. Eval. Rev. **41**(4) (April 2014) 70–73

[33] Trevisan, M., Finamore, A., Mellia, M., Munafo, M., Rossi, D.: Traffic analysis with off-the-shelf hardware: Challenges and lessons learned. IEEE Communications Magazine **55**(3) (March 2017) 163–169

[34] Vassio, L., Drago, I., Mellia, M.: Detecting user actions from http traces: Toward an automatic approach. In: 2016 International Wireless Communications and Mobile Computing Conference (IWCMC). (Sept 2016) 50–55