LENTA: Longitudinal Exploration for Network Traffic Analysis from Passive Data

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2927409, IEEE Transactions on Network and Service Management



# LENTA: Longitudinal Exploration for Network Traffic Analysis from Passive Data

Andrea Morichetta *Student Member, IEEE*, Marco Mellia, *Senior Member, IEEE*

***Abstract*—In this work, we present LENTA (Longitudinal Exploration for Network Traffic Analysis), a system that supports the network analysts in the identification of traffic generated by services and applications running on the web. In the case of URLs observed in operative network, LENTA simplifies the analyst's job by letting her observe few hundreds of clusters instead of the original hundred thousands of single URLs. We implement a self-learning methodology, where the system grows its knowledge, which is used in turn to automatically associate traffic to previously observed services, and identify new traffic generated by possibly suspicious applications. This approach lets the analysts easily observe changes in network traffic, identify new services, and unexpected activities.**

**We follow a data-driven approach and run LENTA on traces collected both in ISP networks and directly on hosts via proxies. We analyze traffic in batches of 24-hours worth of traffic. Big data solutions are used to enable horizontal scalability and meet performance requirements. We show that LENTA allows the analyst to clearly understand which services are running on their network, possibly highlighting malicious traffic and changes over time, greatly simplifying the view and understanding of the network traffic.**

***Index Terms*—Big data, Clustering, Edit Distance, Machine Learning, Security, Traffic Monitoring**

## I. Introduction

In the 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 [36]. Video streaming, music, VoIP, chat, and web services today run on the top of HTTP or HTTPS. Even malware prefers HTTP to let infected clients communicate with Command and Control (C&C) servers [2].

While this has simplified the structure of the protocol stack, the complexity of current developments raveled the analysis of web traffic, so that it is tough to understand which services are running on the network. Fig. 1 gives the intuition of the variety of traffic today, reporting the growth in the number of unique URLs, we measured them observing the Internet activity of hundreds of users in a real network. Data refers to March 2016, where still more than 40% of traffic was carried by HTTP [23], [42]. As the Figure shows, the users access every hour several tens of thousands unique URLs (solid curve - left y-axis) via HTTP, with the total number of unique URLs (dotted curve - right y-axis) that grows to more than 430 000 URLs after one week. In a corporate scenario, the network security analyst is interested in periodically processing traffic to observe which services are accessed by terminals, to then take informed actions in case some anomaly is present. This task requires to process a consistent amount of traffic so to guarantee the correlation and comparison between events that a too coarse analysis would miss. This scenario calls for the support of automatic tools to process, analyze, and extract useful information from the raw data, *i.e.*, a big data solution.

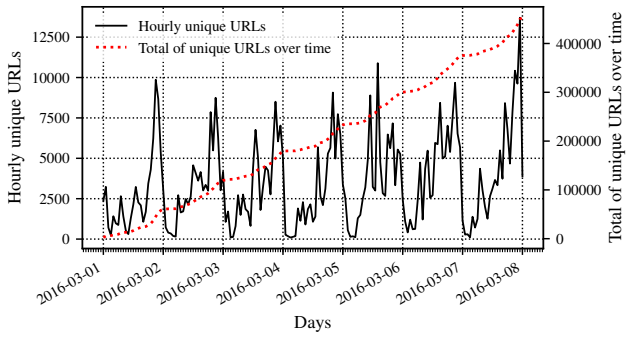



Figure 1: Evolution of unique URLs observed on the ISP network.

In this context, other big data approaches started to emerge to scale the analysis of traces [5], [6], [17], [26], [40]. They offer the ability to process massive data [26], and run machine learning methodologies for traffic classification [17], [40], traffic monitoring analytics [6], or in general to support the so-called data science process, *i.e.*, the extraction of insights from massive data [5]. For the latter case, unsupervised machine learning, *i.e.*, clustering algorithms [1], allows one to reduce the size of the problem from a hundred thousand single objects – the unique URLs – to few hundreds of clusters, containing "similar" URLs. Notice that most URLs carried by a network do not derive from an intentional user action (*e.g.*, the click of a link on a page), but are instead due to applications fetching objects (*e.g.*, elements in a web page, or system component for a web-app) [43]. These latter groups often have 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 is not trivial to define.

Our proposed solution LENTA leverages a novel clustering algorithm that enhances classic clustering algorithms by simplifying the parameter choice, an often cumbersome process. We call it Iterative DBSCAN, or IDBSCAN for short. When clustering URLs, it outperforms other off-the-shelf clustering algorithms in grouping URLs with a similar structure. The

The research leading to these results has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-129, "BigDAMA" and with the SmartData@PoliTO center for data science and big data. A preliminary version of this work has been presented to ITC 30 [33].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2927409, IEEE Transactions on Network and Service Management



analyst can then examine URLs in each cluster to identify the service that generated them, i.e., giving them a possible label.

Next, we design a self-learning approach that lets the system build system knowledge. LENTA compares newly found clusters to those found in the past, so to automatically re-assign the same label if already known. In this way, LENTA offers the analyst only previously unseen clusters, while known traffic is automatically labeled. This process lets the analyst highlight changes and the birth of previously unseen traffic, building a longitudinal view.

We test LENTA on two real use cases. In the first, a passive probe observes thousands of users in an ISP network, for three weeks. In the second case, we use a Man In The Middle (MITM) proxy to collect and process all traffic coming from single hosts. Results show LENTA (i) ability in aggregating thousands of URLs into few clusters, which are easy to investigate and associate to services or malicious activities; and (ii) the capability of identifying new traffic generated by previously unknown applications.

This paper extends our prior works. In [32], we first proposed to use clustering for highlighting and standing out groups of URLs; in [33], we introduced IDBSCAN and the self-learning methodology; finally, in [13] we addressed the scalability of the clustering step in Spark platform. Compared to our previous works, we offer in this paper a systematic evaluation of all LENTA components by comparing IDBSCAN results versus other clustering algorithms. We introduce the concept of eviction in the self-learning stage and offer an analysis of trade-off between completeness and performance. Furthermore, we discuss and optimize the system implementation by using a tree structure to reduce the lookup time and allow horizontal scalability. Considering experiments, we broaden the analysis of the HTTP trace to three weeks to have a more clear view on the evolution of the system over time. We consider a second use case, with a novel dataset of HTTPS traces where we analyze the evolution of web usage of single users.

The rest of the paper is organized as follows. Section II position our work in the context of previous research. Section III gives an overview of the rationale of our work and examines, from a broad perspective, the general structure of LENTA. Section IV describes the steps of the methodology. Sections V and VI explain and justify the choices of our solution, and detail results considering real case scenarios. Section VII provides an insight into the implementation of LENTA and Section VIII concludes the paper.

## II. Related Work

Thanks to the complexity and richness of network traffic, the last decade has witnessed several research studies concerning the use of machine learning techniques to automatically extract information. The critical applications are traffic classification and anomaly detection. In both cases, the application of clustering technique is of great interest.

Authors of [18], [19] 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, considering all the features. The latter uses a hierarchical clustering technique to merge similar bags of bi-grams, extracted from messages collected from Internet Relay Chat monitoring. Conversely to our solution, they focus uniquely on a specific target, *i.e.*, botnets, and they use different features and techniques for clustering.

Considering traffic classification, Erman et al. [11] use transport layer statistics and test clustering algorithms (namely, k-Means, DBSCAN, and AutoClass) over different labeled datasets. Authors of [27] use labeled traffic flows and k-Means clustering for classification of TCP flows. Wright et al. in [45] leveraged k-Means over Hidden Markov Models of Client-Server and Server-Client communications as an intermediate step toward the detection of applications behavior over encrypted traffic. The goal is, again, traffic classification. In this paper, we focus on HTTP traffic and extract clusters looking at the structure of the URLs.

Some studies focused on text and string mining techniques, with the goal of clustering network traffic. In the field of network security, authors of [35] use a two-level clustering process, leveraging the single-linkage hierarchical algorithm to disclose similarities between malicious URL; Levenshtein distance, together with Jaccard Index, is used in the second clustering stage. They target malware signature building explicitly. In [24], semantic features of the URLs are used to target the same problem, using DBSCAN and Jaro-Wrinkler distance. Authors of [30] use the Levenshtein distance aiming at detecting phishing sites, whose names are built using typical spelling mistakes. Gao et al. [15] use clustering techniques to detect spam campaigns on Facebook, looking at similarities in destination URLs. Other works target YouTube traffic [16], or P2P traffic [5], and use features extracted from TCP flows like round trip time, data exchanged, and other domain-specific metrics.

All these works focus on a specific class of traffic, with a specific goal. Here we aim at broadly exploring HTTP traffic, in general. We focus on URLs, which are strings, for which defining a distance and a clustering requires ingenuity. LENTA is a general methodology that examines URLs from HTTP traffic to group elements that look similar. This approach allows the analyst to quickly identify patterns, anomalies, and novelties in traffic, services, and users behaviors. The system we propose consider all HTTP traffic, and not just malicious URL or URL generated by malware during their activity.

From a system point of view, we address the engineering and deployment of scalable clustering algorithms, in particular for density-based algorithms. We adopt big data approaches for which ingenuity is required to parallelize the execution. In the system community several authors are working on scalable clustering solutions [10], [20], [29], [22]. They mostly take advantage of feature space partition, leveraging Spark, and MapReduce paradigm. These approaches, however, are confined to the class of problems where Euclidean distance metrics can be used and provide approximated clustering. Recently, Lulli et al. in their work NG-DBSCAN [28], propose a methodology to overcome those limitations that consist in an approximated version of DBSCAN, based on a *vertex-*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2927409, IEEE Transactions on Network and Service Management



*centring* programming paradigm, built on the concept of graphs. It provides an increase in terms of performance and scalability at the cost of lower accuracy. Other works realize parallel optimizations of density based algorithms, focusing on GPU computing capabilities [44], or the multiprocessing API OpenMP in conjunction to graph techniques [34].

Our approach follows a different angle and targets the parallelization of the distance matrix computation. This strategy stems from the fact that the computation of string similarity is per se a resource-demanding job. Being URLs possibly very long strings, this poses severe scalability issues that we solve by providing a map-reduce solution. Once LENTA extracts the distance matrix, we run DBSCAN in a centralized manner, thus not facing any approximation.

## III. Motivation and System overview

In this paper, we target the analysis of passively collected HTTP traffic, which still today amounts to more than 40% of web traffic [23], [42]. While the majority of malicious traffic still runs on top of HTTP [2], services are moving on HTTPS. In order to observe HTTPS traffic too, several solutions are at the disposal of network and security analysts, like MITM solutions, also known as SSL forward proxies, suitable for corporate scenarios.[1, 2]

### A. Motivation

Our goal is to group URLs based on their similarity. We choose to leverage string distance to generate homogeneous groups of URLs instead of just merging those elements that have, *e.g.*, a common domain name. In principle, we strive for grouping together all those URLs that refer to the same service while separating URLs of different services. We present some real cases to give the reader the intuition (and the complexity) of doing this. Tab. I shows examples of URLs. *A1*, *A2*, and *A3* belong to the same malware called TidServ – that we spotted in our dataset through a professional IDS provided to us by a leading cybersecurity company. All URLs share substrings in the object path, but with strictly different domain names and URLs. This is a common behavior in malicious applications which apply approaches to change the domain name rapidly, with the goal of evading static blacklist-based controls, the so-called DGA (Domain Generation Algorithm) technique, successfully used by several malware programs like *Conficker* [37] and *Torpig* [39], and addressed in various research works [4], [7]. *B1* and *B2* illustrate two URLs generated by Sony connected Smart-TVs which access the same service, but with different URLs. This characteristic is representative of services that employ the same web platform, and that can be interesting to point out. In both the above examples, we would like the algorithm to form two groups, one for the malware, one for Smart-TV traffic.

We remark that grouping by domain name is not enough. Indeed, certain domains host logically very different services.

[1]https://www.paloaltonetworks.com/documentation/71/pan-os/pan-os/decryption/ssl-forward-proxy

[2]https://www.juniper.net/documentation/en_US/junos-space15.2/topics/concept/junos-space-ssl-forward-proxy-overview.html

Table I: Examples of similar URLs

| | |
|---|---|
| swltcho81.com/[...]VyPTQuMCZiaWQ9[...] | A1 |
| rammyjuke.com/[...]VyPTQuMCZiaWQ9[...] | A2 |
| iau71nag001.com/[...]VyPTQuMiZiaWQ9[...] | A3 |
| bravia.dl.playstation.net/bravia/WidgetBundles/[...]/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 |

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. To reach this goal, we use clustering approaches.

### B. System Overview

Fig.2 sketches the overall process. We process URLs in batches, each one marked as $UG(i)$, where we insert all unique URLs seen during the *i-th* time interval of duration $\Delta T$. In our analysis, we solely consider unique URLs, since our goal is to understand which resources are fetched by clients, independently of their popularity. At the end of a period, collected URLs are clustered in $\mathcal{C}(i)$. Several challenges arise here, from the computation of the similarity between two URLs to the proper choice of the clustering algorithm, from the parameter settings to a scalable design.

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

At last, we compare clusters found in the current batch with those found in the past, which we collect in a structure that we name System Knowledge and for which we use the notation $\hat{\mathcal{Z}}(i-1)$. If there is no match, then the current cluster is considered new and added to the System Knowledge after eventually the analyst's inspection, to provide a meaningful label. As we will show, the availability of several URLs of the same type substantially simplifies the labeling process.

## IV. Methodology

Here we detail our methodology, defining the techniques adopted at each step. We start describing the process of data collection. Then we analyze how to compute distance measures from data. Subsequently, we describe the clustering algorithm and illustrate the sampling technique. Finally, we detail operations correlated to the update and support of the System Knowledge.

### A. URL Extraction

The extraction of URLs from web traffic is the first step of the process. Visibility in live traffic can be obtained using a

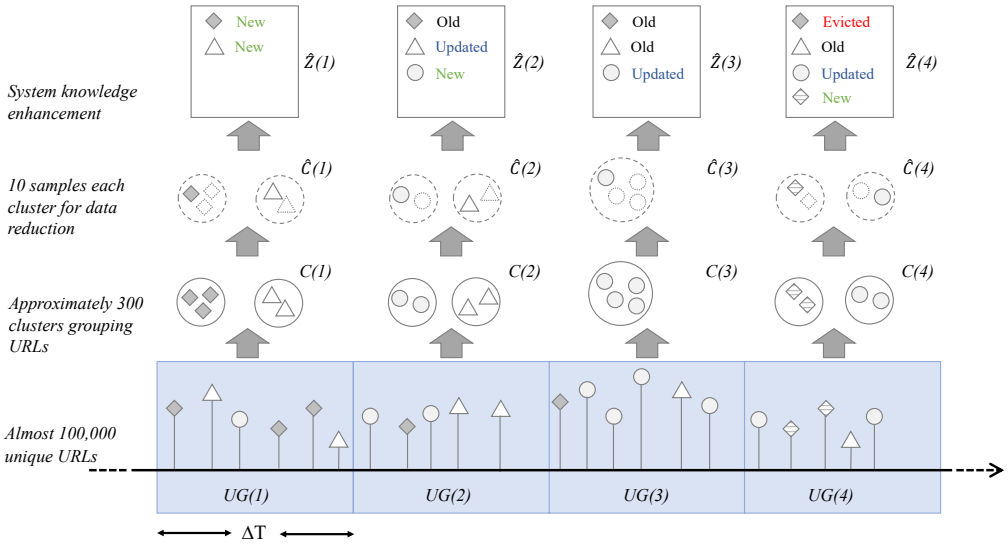



Figure 2: LENTA overview. From the bottom, URLs are grouped in batches to then extract clusters. The clusters are sampled and used to update the System Knowledge.

passive sniffer, or a proxy, which, in case of a MITM proxy, would allow the processing of HTTPS traffic too. In this work, we rely on both approaches: Tstat [41], a scalable passive sniffer to monitor high-speed links; and ERMES proxy[3], a MITM solution we engineered to observe also HTTPS traffic.

Tstat implements an efficient Deep Packet Inspection (DPI) architecture that logs HTTP requests observed in packets. We have been running it in collaboration with an ISP for more than five years [42]. For the experiment in this paper, we use a three-week-long HTTP trace collected in March 2016 in the ISP network, where traffic from more than 20 000 customers was visible. To protect users privacy, we removed all parameters in the URL, and only saved unique URLs.[4] As shown in Fig. 1, we observe more than 430 000 unique URLs during a week, more than 60 000 per day.

ERMES proxy is a software module that runs on end hosts. By installing a trusted key, and configuring a system-wide proxy, it gains visibility on all HTTP and HTTPS requests. ERMES proxy logs all entries and uploads them to a centralized repository in our campus. We asked volunteers to install the ERMES proxy for at least one month. We recruited participants on social networks, specialized forum, and among students in our University. We provided monetary incentives and involved them in an experiment aiming at showing them the pervasiveness and danger of web tracking. The volunteers have explicitly approved our data collection program, and the project was also subject to a privacy impact assessment that we redacted together with the data protection officer of our institution.

Given a log of URLs, we form and analyze, after every period of duration $\Delta T$, a URL group $UG(i)$. In our experiments, we choose $\Delta T = 24h$. The daily periodicity of traffic - which reflects the typical daily activity of users - justifies this (see Fig. 1).

When using Tstat, we consider all unique URLs generated by hosts in the monitored network, mimicking the case of the network analyst that is interested in observing what the network carries. Instead, traces collected by ERMES-proxy are processed considering all URLs generated by every single device, mimicking the case of the security analyst that has access to HTTPS traffic and is interested in each single device tracing.

### B. Distance Definition

Clustering is the task of grouping a set of objects in such a way that the ones in the same group are more similar to each other than to those in other groups.

In our case, objects are URLs, *i.e.*, strings, for which there is no well-accepted notion of distance. We focus on a particular class of similarity metric, the edit-distance [9]. 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 $s_2$. For this purpose, we propose a custom modification of the Levenshtein distance, $d_{LVS}$ [25]. In detail, we count the total number of insertions and deletions and weight each replacement by two. The rationale is that a replacement corresponds to one combined operation of deletion and insertion. Given the peculiarity of URLs, whose length may vary widely, we normalize the results in a $[0, 1]$ range by dividing by the sum of string lengths,

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

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

[3]https://www.ermes.polito.it

[4]The usage of this data set has been discussed and approved by our institution's ethics committee, and by the ISP security group.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2927409, IEEE Transactions on Network and Service Management



different. In [32] we demonstrated that this distance definition performs better than other off-the-shelf solutions.

### C. Clustering

In the field of unsupervised methods, clustering is one of the most popular. 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, they generally work with wide 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 left out from the final clusters; this is a particularly interesting feature, since those points may be useful for anomaly detection.

*1) DBSCAN:* We built upon and improved the well-known DBSCAN algorithm [1]. DBSCAN identifies a cluster as the concatenation of successive dense areas in the data space. Given an object *o*, it is possible to measure its density by considering the number of elements close to it. DBSCAN finds the core points, *i.e.*, those objects that have dense neighborhoods; then it connects these core points and their neighbors to form the dense regions, *i.e.*, the clusters. The $\epsilon$ parameter defines the neighborhood area. This parameter 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 $\epsilon$.

*2) Iterative DBSCAN (IDBSCAN):* The setting of the MinPoints and $\epsilon$ parameters is in general difficult. In particular, MinPoints can be reasonably set using domain knowledge since it represents the minimum number of elements of a cluster. $\epsilon$ is instead hard to set, especially if the used distance is not well known. In the first version of our work, CLUE [32], we had to set $\epsilon$ by manually tuning it, a cumbersome and error-prone task. Here we propose a new approach to automatically compute $\epsilon$, while also improving the final clustering. The intuition is to iteratively run DBSCAN, each time using a different value for $\epsilon$, and each time accepting only those clusters that are well-shaped. Objects in bad-shaped clusters are eventually re-clustered in the next iteration, with a different choice of $\epsilon$. This approach produces a remarkable improvement of LENTA's clustering stage, by further splitting/merging clusters at each iteration, until they eventually form a well-shaped cluster. 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 the system would ignore.

We define $\epsilon$ by using an a priori rule, *i.e.*, we want the algorithm to cluster a given percentage $\eta$ of objects at each iteration. To choose the proper $\epsilon$ that would guarantee this, we rely on the $k$-Distance graph rule [1]. Let $k =$ MinPoints. For each object $i = 1, \ldots, N$ in the current batch, 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 $\eta = 75\%$ of points. We set $\epsilon = d_{th}$. With this choice, 75% of objects have at least $k =$ MinPoints objects at a distance smaller than $\epsilon$. 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 [38]. 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:

$$a(i) = \frac{1}{\|C\|} \sum_{j \in C \neq i} d_{URL}(i,j) \tag{2}$$

$$b(i) = \min_{C' \neq C} \left( \frac{1}{\|C'\|} \sum_{j \in C'} d_{URL}(i,j) \right) \tag{3}$$

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{4}$$

It results $s(i) \in [-1, 1]$. 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 the clustering process might have assigned $i$ to the wrong cluster. The average $S(C) = E[s(i), i \in C]$ overall 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 $S(C) > S_{min}$. If $C$ is well-shaped, we insert $C$ in the set of clusters found so far. Otherwise, we put back all points in $C$ in the set of points that we would consider for the next iteration. By setting a maximum number of possible iterations, we avoid dead loops. If the IDBSCAN process is not able to rearrange a cluster, the algorithm labels the contained elements as noise.

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

$$\mathcal{C} = \bigcup_j \{C_j | S(C_j) > S_{min}\} \tag{5}$$

We ran several experiments to check the quality of clustering for different values of $S_{min}$ and $\eta$. In a nutshell, the algorithm is robust to the choice of $\eta$, while any value of $S_{min} > 0$ gives good results. For the sake of brevity, we do not report outcomes here. Our choice of $\eta = 75\%$ and $S_{min} = 0.3$ is conservative and produces very well-shaped clusters.

### D. Sampling for Data Reduction

Once we obtain the final clusters, 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.

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2927409, IEEE Transactions on Network and Service Management



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 $\hat{C}_j = sample(C_j, m)$.[5]

$sample(C_j, m)$ 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 URLs 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 URL $s_i \in C_j$

$$\left\{E_{s_k \in C_j}[d_{URL}(s_i, s_k)], \forall s_i \in C_j\right\} \quad (6)$$

The selected elements are the ones that 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 samples that includes both elements 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.[6] The medoid is generally an appropriate choice to describe a group of elements, but it is more appropriate for spherical and homogeneous clusters. Since a cluster in IDBSCAN consists of a chain of interconnected smaller spherical dense areas, the choice of only one point would exclude other possibly distinguishing instances. In this sense, the percentile sampling produces a sampling that better represents the population of the cluster.

### E. System Knowledge Enhancement Intuition

LENTA maintains the set of clusters found in the past in the System Knowledge $\hat{\mathcal{Z}}(t)$. 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

$$d_{min}(\hat{C}, \hat{\mathcal{Z}}) = \min_{\hat{Z} \in \hat{\mathcal{Z}}} \left( d\left(\hat{C}, \hat{Z}\right)\right) \quad \text{where } d(\hat{C}, \hat{Z}) = \min_{\substack{c \in \hat{C} \\ z \in \hat{Z}}} d_{URL}(c, z) \quad (7)$$

Let $\hat{\mathcal{C}}(t)$ be the result of the clustering of the current batch. We need to check if a cluster $\hat{C}_j(t) \in \hat{\mathcal{C}}(t)$ contains the similar content of an already registered one, 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

$$\hat{Z}_l(t-1) = \arg\min \left( d_{min}\left(\hat{C}_j(t), \hat{\mathcal{Z}}(t-1)\right)\right) \quad (8)$$

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

$$\hat{\mathcal{Z}}(t) = \hat{\mathcal{Z}}(t-1) \cup \left\{ \hat{C}_j(t) \in \mathcal{C}(t) \mid d_{min}\left(\hat{C}_j(t), \hat{\mathcal{Z}}(t-1)\right) \geq \alpha \right\} \quad (9)$$

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

### F. 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 with the old one and may contain new knowledge, *e.g.*, some important changes in the particular service or differences in the structure or information carried by URLs. It is vital to register, if possible, those updates.

We apply 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,

$$z_i := c_i \leftarrow p \ \ \forall i \in [1, m], \ \ z_i \in \hat{Z}_l(t-1), c_i \in \hat{C}_j(t) \quad (10)$$

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

### G. Pruning based on Inactivity

The System Knowledge keeps information on clusters since the first time we had seen them. Let $t_{init}$ be the time in which we added it to $\hat{Z}$, and $t_{last}$ be the last time we encountered it, textiti.e., when the system associates a new cluster to it.

We implement a pruning mechanism to remove those inactive clusters, that contain content that users did not visit in the recent past. We use $t_{last}$ to remove old and inactive clusters. Given $\hat{Z}$, we define $\Delta T_{inactive} = t - t_{last}$. If $\Delta T_{inactive} \geq \Delta I_{thresh}$, with $\Delta I_{thresh}$ defined as *Inactivity Threshold*, then $\hat{Z}_{inactive}$ is removed from $\hat{\mathcal{Z}}$. We discuss the impact of $\Delta I_{thresh}$ in Section VI.

## V. Results

In this section, we report the results obtained by applying LENTA. Our objective is to analyze the different components of the system in order to verify the satisfaction of the main goals of LENTA.

In order to achieve that, we first compare IDBSCAN with other off-the-shelf density-based solutions. For this, we run a set of experiments over the unique URLs extracted in the first 24 hours of data collection, and we analyze the resulting clusters.

### A. Clustering Algorithm Analysis

*1) Other Density-Based Algorithms:* To have a broader perspective on density based algorithms and on solutions which claim to overcome the complex tuning of the $\epsilon$ parameter, we consider the following well-known density-based clustering algorithms.

*a) OPTICS (Ordering Points To Identify the Clustering Structure) [3]:* Ankerst et al. addressed the difficulty of setting DBSCAN parameters and, in particular, the choice of $\epsilon$. 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. So, the contribution is to compute core points for high-density areas first, finding in that way the denser clusters. OPTICS order points according

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

[6] The medoid is different from the centroid since the first consists in selecting an element among the ones of the cluster, while the second does not have this restriction.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2927409, IEEE Transactions on Network and Service Management



to their density and provides that list in written or graphical form. Clusters can be identified graphically or automatically from that structure, and so different local densities, *i.e.*, $\epsilon$, may be defined in order to extract clusters in different areas of the data space.

*b) HDBSCAN (Hierarchical DBSCAN) [8], [31]:* A limitation of DBSCAN is that it is not able to identify clusters made of points that lay at different densities. HDBSCAN aims at solving this problem. It first creates a tree representation of all the possible clusters for different $\epsilon$. 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 [21], is maximized. Thanks to this approach, there is no need to tune the $\epsilon$ parameter.

*c) CANF (Clustering and Anomaly detection method using Nearest and Farthest neighbor) [12]:* This method finds the nearest and furthest neighbors to define subgroups of data. In creating the subgroups, it does not need to consider global parameters like $\epsilon$. It computes the radius of subgroups based on the variance of data points, and the number of points in each subgroup and its volume are adaptive to data distribution. So it can identify clusters with different shape and densities. Since CANF uses subsampling, the time complexity is reduced compared to the methods that use the aggregate data set. However, being iterative, worst case complexity entails the comparison of all distance pair again.

*2) Performance Comparison:* We compute the quality of clustering considering the first day of traffic collected by Tstat, containing 59 543 unique URLs. We test all algorithms setting the value of $MinPoints = 20$; $\epsilon$, used by DBSCAN and OPTICS, is set at the value $\epsilon = 0.4$, as suggested in [33]. Off the shelf Scikit-Learn implementations of DBSCAN and HDBSCAN are used, OPTICS is executed using the pyclustering version, IDBSCAN uses Scikit-Learn DBSCAN as the building block, while CANF uses authors' developed code.

To evaluate the performance of the algorithms, we report statistics about the clustering results in terms of clusters quality, measured using the silhouette coefficient. Fig. 3 depicts the silhouette distribution among all clusters, for each algorithm. Recall that silhouette smaller than 0 is an indication of lousy clustering. Inspecting the boxplots, IDBSCAN shows the best distribution of silhouette, HDBSCAN the worst. This behavior is thanks to the fact that IDBSCAN rejects by construction all clusters whose silhouette is smaller than $S_{min}$. Moreover, the IDBSCAN iterative process splits re-clusters poorly shaped clusters with variable $\epsilon$. By contrast, the other algorithms produce considerably more variability in silhouette values, since they do not discard poorly shaped clusters.

Tab. II provides more details, complementing the silhouette coefficient metric. The results confirm that IDBSCAN appears to be the best algorithm in term of percentage of URLs clustered, with CANF being the worst. Comparable results are obtainable with HDBSCAN, which, however, generates more clusters, thus increasing the analyst work during the inspection phase. The other algorithms generate large clusters of more than 15 000 elements (more than 25% of the data set size), which are difficult to analyze and usually aggregate

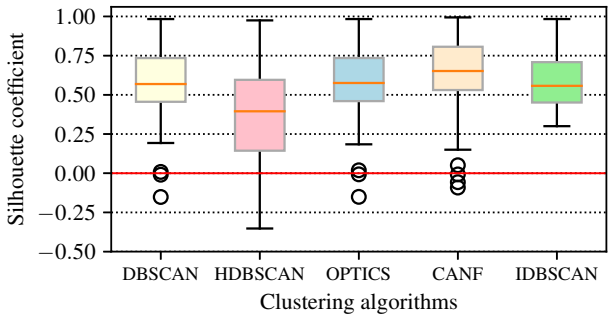



Figure 3: Boxplot representing the mean silhouette values for the clusters obtained by each different algorithm on the first day of traffic.

diverse URL types (as also seen by the low silhouette values). The comparison between clusters' statistics and the silhouette coefficient results in Fig. 3 helps to shed light on clustering performances. Specifically, the first, second (median) and third quartile of cluster size they provide clear information on clusters structures. Even if CANF and DBSCAN have a good median silhouette value, they form a large number of small clusters, with the first quartile equal to five. Small clusters are usually denser, influencing thus the silhouette statistics.

Considering execution time, IDBSCAN is slower than DBSCAN and HDBSCAN because of the iterations. Still, the clustering is completed in less than 850 s. In Sec.VII we provide more details on scalability.

### B. Malware Detection - TidServ Use Case

We now test the clustering stage over a dataset including one day of HTTP traffic from 34 hosts, 14 of which are flagged by the IDS as infected by the TidServ malware previously described. We randomly select the other 20 hosts from the population. Overall, we obtained 78421 unique URLs, from which only 228 represent, according to the IDS, TidServ traffic.

We run IDBSCAN over all URLs and check those clusters which contain at least one TidServ URL. In total, IDBSCAN identifies 7 clusters according to the distribution reported in Tab.III. Each cluster contains URLs not originally flagged by the IDS. By manually checking those, we confirm that IDBSCAN correctly assigns those to TidServ clusters, being those very likely to be false negatives for the IDS (i.e., the IDS did not flag those despite being malicious). Table IV details cluster 7. In this case, only the first (in bold) URL out of 37 is flagged as malicious by the IDS. The similarity within all URLs is however clear, letting us conclude that those are false negatives for the IDS. This clear example explicates how LENTA could be used to support the generation and update process for IDS signatures.

### C. Clustering Analysis and Labeling

In this section, we provide experimental results. We choose $\Delta T = 24\,\text{h}$, $\eta = 0.75$, $S_{min} = 0.3$, $p = 0.2$ and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2927409, IEEE Transactions on Network and Service Management



Table II: Clustering results obtained applying different density-based algorithms over one day of traffic.

| Algorithm | Percentage clustered ($S(C) \geq S_{min}$) | N. clusters | Size largest cluster | $S(C)$ largest cluster | Size smallest cluster | $S(C)$ smallest cluster | Mean cluster size | 25% cluster size | 50% cluster size | 75% cluster size | Computational Time (s) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| DBSCAN | 45.14 | 238 | 15246 | -0.15 | 16 | 0.41 | 148.28 | 5.0 | 16.5 | 57.75 | 113.70 |
| HDBSCAN | 53.16 | 563 | 4360 | 0.52 | 20 | -0.17 | 82.34 | 28.0 | 41.0 | 65.0 | 218.43 |
| OPTICS | 44.65 | 227 | 15214 | -0.15 | 2 | 0.44 | 205.45 | 27.0 | 44.0 | 89.0 | 8175.82 |
| CANF | 29.67 | 233 | 15946 | -0.09 | 2 | 0.84 | 148.28 | 5.0 | 16.5 | 57.75 | 1500.73 |
| **IDBSCAN** | **55.55** | **283** | **4359** | **0.52** | **12** | **0.41** | **147.87** | **27.0** | **44.0** | **83.0** | **843.63** |

Table III: TidServ clusters identified by IDBSCAN.

| ID | Tot. URLs | TidServ URLs | Hostnames | Most common hostname |
|---|---|---|---|---|
| 1 | 192 | 118 | 14 | 81hja01aala.com |
| 2 | 79 | 75 | 1 | wuptywcj.cn |
| 3 | 32 | 18 | 2 | clickpixelabn.com |
| 4 | 6 | 6 | 1 | biiwf3iidpkxiwzqmj.com |
| 5 | 6 | 5 | 1 | zl091kha644.com |
| 6 | 5 | 5 | 1 | zhakazth.cn |
| 7 | 37 | 1 | 3 | lkckclckl1i1i.com |

Table IV: URLs in cluster 7. The IDS flagged only the first.

**gnu4oke0r.com/4...PTQuMCZiaWQ9NWJjNWFiMjE1Yj...5pdCZxPWxvdWlzIGNydWlzZXM=16h**
lkckclckliii1i.com/...PTIuNCZiaWQ9NWJjNWFiMjE1Yj...jE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9MA==27g
lkckclckl1i1i.com/...PTIuNCZiaWQ9NWJjNWFiMjE1Yj...jE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9MA==27g
lkckclckliii1i.com/...PTIuNCZiaWQ9NWJjNWFiMjE1Yj...jE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9MA==26g
lkckclckl1i1i.com/...PTIuNCZiaWQ9NWJjNWFiMjE1Yj...jE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9MA==26g
lkckclckliii1i.com/...PTIuNCZiaWQ9NWJjNWFiMjE1Yj...jE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9MA==18x
lkckclckl1i1i.com/...PTIuNCZiaWQ9NWJjNWFiMjE1Yj...jE1YyZhaWQ9MzAwMDEmc2lkPTAmcmQ9MA==18x
etc.

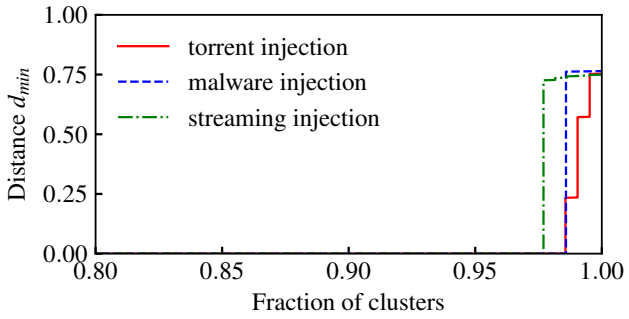



Figure 4: Curves of distances when new traffic is injected in the controlled experiment. Top 20% clusters are reported.

$MinPoints = 20$ to look for well-shaped and big enough clusters. We tested different parameters, observing little changes. Experiments are not reported here for the sake of brevity.

We start to analyze the first day of traffic. As seen from Tab. II, LENTA obtains 283 clusters from the set of 59 543 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 result very well shaped.

The top part of Tab. V shows the most massive clusters, while the bottom part displays those with the highest silhouette. The table reports the silhouette $S(C)$, the most common hostname in the cluster (in brackets the total number of distinct 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 different hostnames. That behavior is not to be taken for granted, as often the complexity of URLs structure tend to increment the distance also for actually similar elements.

After this stage is already possible to identify some suspicious clusters, for instance, 30 unique URLs form a cluster where URLs have all the same IP address `219.129.216.161` – but random paths. After further analysis[7], this cluster is indeed found to be malicious. Other suspicious clusters emerge as well. At last, it is essential 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` finds its representation into two clusters, one serving images (.GIF), and the other exchanging control information like devices reports.

[7]Google results: https://goo.gl/q3DgT8; VirusTotal results: https://goo.gl/fqrNkG

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

### D. In vitro Experiment

To evaluate the reaction of LENTA concerning 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 can identify the new traffic. Results are reported in Fig. 4, 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 the previous step $t = 0$. We report only the last 20% of clusters, ordered by $d_{min}(\hat{C}, \hat{\mathcal{Z}})$. As clearly shown, LENTA is able to recognize old traffic ($d_{min} = 0$ for those clusters in $UG(t)$ that were already present in $UG(t-1)$. The new traffic is clustered separately, with $d_{min} > 0$. Tab. VI details the results of the experiment. First, new clusters contain only new URLs injected in each step of the process. Second, LENTA identifies multiple new clusters for each stage. This behavior 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). Torrent clusters correspond to different swarms and trackers. Third, $d_{min} > 0.3$ for all

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2927409, IEEE Transactions on Network and Service Management



Table V: 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 |

Table VI: New clusters highlighted during the comparison with the System Knowledge.

| Experiment stage | $\mathbf{d_{min}}$ | Main hostname(s) |
|---|---|---|
| $UG_1$ **Torrent** | 0.75 | i-1006.b-0.ad.bench.utorrent.com, i-1005.b-0.ad.bench.utorrent.com |
| | 0.57 | b.scorecardresearch.com, pixel.quantserve.com |
| | 0.23 | tracker.aletorrenty.pl:2710, torrent.gresille.org |
| $UG_2$ **Malware** | 0.76 | wuptywcj.cn |
| | 0.76 | rlyg0-6nbcv.com,riygo-6nbcv.com, riyg0-6nbcv.com,iau71nag001.com |
| | 0.76 | bangl24nj14.com,switcho81.com, rammyjuke.com,skolewcho.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 |

clusters but one in the Torrent data, for which $d_{min} = 0.23$. This cluster would associate with a previously seen cluster. The association is correct, and URLs have a very similar syntax to the one already found and related to a different tracker service, `tntvillage`.

### *E. System Knowledge Parameter Tuning*

Once clusters are identified, we extract a digest via sampling. This stage is one of the most critical since it is essential to balance the representativeness and the complexity of the System Knowledge. Here we discuss the impact of the parameters related to this step, namely, the sampling methodology $sample(C_j, m)$, the number of samples $m$ to keep, and the threshold $\alpha$ to associate a new cluster with an old one.

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.

To choose which strategy works best, we run an experiment in which we split the clusters in $\mathcal{C}(0)$ into two sets: the first set builds the System Knowledge $\mathcal{Z}(0)$ and contains half clusters selected at random from $\mathcal{C}(0)$. The second set $\mathcal{C}(1) = \mathcal{C}(0)$ contains all clusters. We apply sampling and compare $\hat{\mathcal{C}}(1)$ is compared to $\hat{\mathcal{Z}}(0)$. We expect the System Knowledge algorithm to identify half of the clusters as already known, and the other half as new.

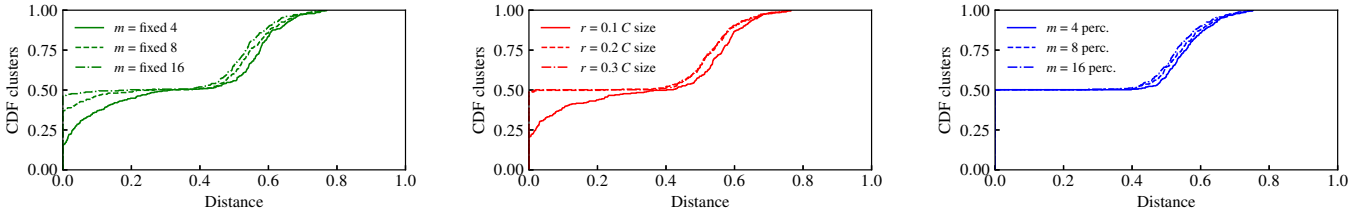
Results are depicted in the plots of Fig. 5 which show $d_{min}(\hat{C}_j(1), \hat{\mathcal{Z}}(0))$, in increasing order, respectively comparing results for $m = \{4, 8, 16\}$ (Fig. 5a - fixed sampling), $r = 0.1, 0.2, 0.3$ (Fig. 5b - proportional sampling) and for $m = \{4, 8, 16\}$ (Fig. 5c - percentile sampling).

We would expect to see an approximation of a step curve, where the first half of the distances are equal to 0 because the comparison entails the same clusters; the second half of the distances should have a value larger than 0 – the higher, the better. Fig. 5 clearly shows that, in case of random sampling, the higher the number of samples, the more visible is the ideal step-curve-behavior. The approximation is excellent, picking a fixed $m$ equal to 16, and very similar to the step curve with proportional $r$ of 20% or 30%.

The situation improves when using percentiles, whose smart sampling guarantees the best results. Indeed, when we consider the percentile, we always obtain a perfect distance of 0 for the clusters that contain the same elements. That is happening because two sets are equal and we deterministically select the representatives.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2927409, IEEE Transactions on Network and Service Management






(a) Fixed sampling approach. (b) Proportional sampling approach. (c) Percentile sampling approach.

Figure 5: $d_{min}$ when 50% of traffic is the same and 50% is new. Different choices of sampling approaches.

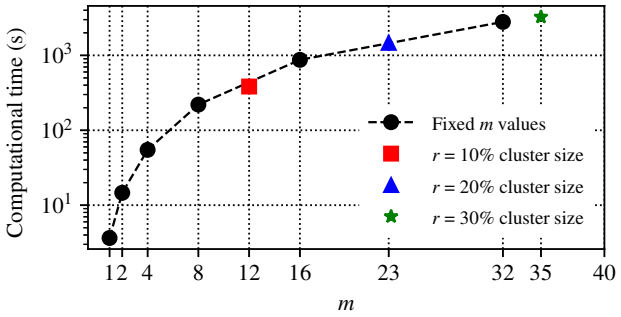



Figure 6: Computation time for different sampling strategies.

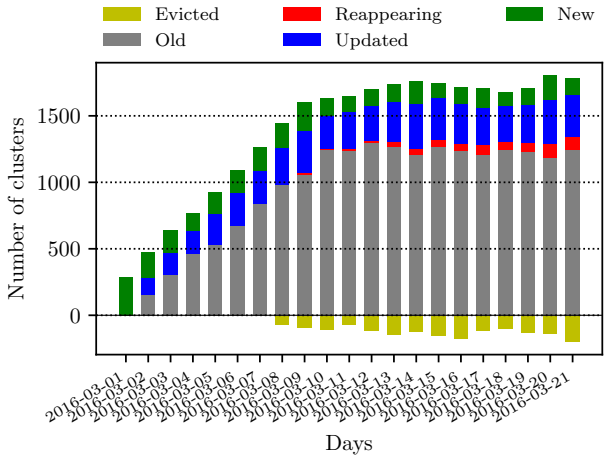



Figure 7: Daily enhancement of system knowledge for HTTP data.

To consider the content of a cluster as belonging to a previously detected entity, its minimum distance with all clusters in the System Knowledge has to be smaller than the threshold $\alpha$. Fig. 5a, Fig. 5b and Fig. 5c clearly show that the new clusters tend to be very dissimilar from the old ones. 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 $\alpha = 0.3$.

As a drawback, increasing $m$ increases the computation complexity, due to the need to compute $O(m^2)$ $d_{URL}(.)$. Fig. 6 shows the experimental computational time using lin/log scales. For variable fraction $r$, we report results using as $x$ value the average number of elements in the clusters. As expected, the curve grows quadratically for $m$ (logarithmically in log scale), with $m$ = 32 and $r$ = 20% or 30% that already have an execution time larger than 3 000 s. Considering the System Knowledge would have thousands of clusters, the best trade-off between cluster similarity identification and computational time is obtained using a fixed $m = 8$. This choice allows System Knowledge to retain enough samples and, simultaneously, it enables accurate matching (see Fig. 5c).

## VI. 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. We then test LENTA over two weeks of HTTPS traffic.

### A. Real Case Scenario

Figure 7 reports the process of System Knowledge evolution over the 21 days under analysis. In the first day all the clusters are added, thus being all labeled as new (in green). From the second day, we can notice different operations: (i) some new clusters join the System Knowledge, (ii) the system operates updates on others when finds similar clusters (blue), and (iii) the remaining group (gray) is in idle. The growth of $\hat{\mathcal{Z}}$ continues up to the beginning of the second week when the pruning action - based on inactivity - starts evicting some old, inactive clusters (yellow). The red bar details how many discarded clusters reappear in the future steps.

The values obtained for the last two features depend on the size of $\Delta I_{thresh}$. To check the impact of this choice, Fig. 8 reports the number of clusters that would be evicted from and eventually re-inserted into system knowledge. As expected the number of both deleted and reappearing clusters decreases enlarging $\Delta I_{thresh}$. This behavior indicates a not negligible periodicity, especially in consecutive days, that may suggest a natural stabilization of the system knowledge size over long periods. For extended analysis, a higher $\Delta I_{thresh}$ may be more suitable, following memory and storage limitations, to let to the system have time to level off.

Compare Fig. 7 with the growth with Fig 1. The number

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2927409, IEEE Transactions on Network and Service Management



Table VII: 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 | 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** |

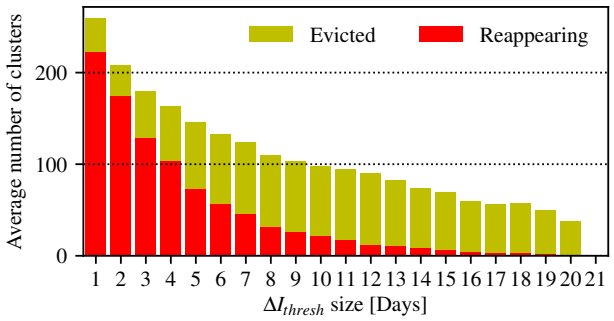



Figure 8: Number of evicted and reappearing clusters given different $\Delta I_{thresh}$.

of unique URLs tops to more than 420 000 after seven days, with on average 72 000 unique URL per day. The number of clusters instead solely reaches 1 600, with less than 200 newly found clusters per day. In a nutshell, LENTA can decrease the amount of information the security analysts have to process by three orders of magnitudes so that they have to inspect less than 200 clusters per day instead of managing several tens of thousands of unique URLs.

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

Tab. VII details the results. In this case, as well, the services are related to streaming, advertising, e-commerce services. Some unexpected 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`, a service not reported in the previous days. URLs are related to a newsfeed specific service. March 4th and 5th, we register some suspicious or malicious traffic. Clusters are related to hijacking services and aggressive advertisement. March 6th is distinctively captivating. Eight out of ten most different clusters contain 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: a rare service like `aww.com.au`, an Australian news website, and webpages translated using the Google Translate online service (curiously translating an adult-content website, possibly to evade content filtering policies).

### B. HTTPS Use Case

Here we report on our second experiment with HTTPS traffic collected via the ERMES proxy.

Differently from the previous analysis, we consider individual users (albeit after anonymization). Starting from the traffic generated each day by each volunteer, we run LENTA first to characterize the usage patterns of each user, and second to observe how it changes over time.

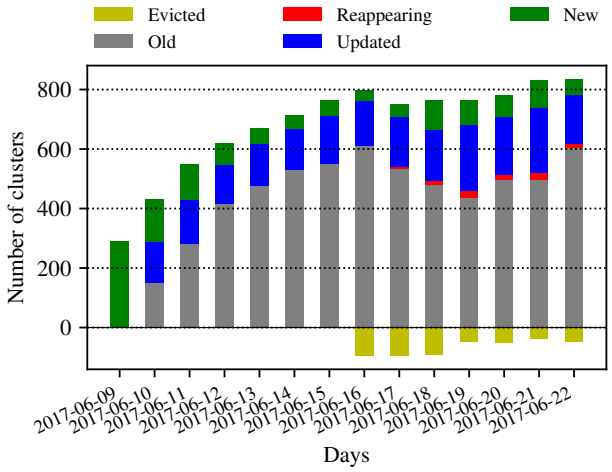



Figure 9: Daily enhancement of system knowledge in HTTPS traces.

Fig. 9 shows the behavior of the System Knowledge for the current investigation. The functioning of LENTA is comparable to the precedent experiment, reported in Fig. 7. However, in this use case, LENTA produces information about the behavior

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2927409, IEEE Transactions on Network and Service Management



of users, showing the flexibility of the system in supporting data exploration. For instance, we report the analysis of the most common categories of services accessed by users. We focus on 171 clusters that resulted in being common for at least two users. The categories extraction derives from manual inspection of clusters. This step results greatly simplified thanks to the similarity and expressiveness of URLs contained in each cluster. Overall, we identified 20 coarse categories of services.

Fig. 10 reports the fraction of clusters that fall into the same category. Almost a quarter of the groups are related to third-party services, which include advertisement, web tracking, and analytics; their pervasiveness affects the results of the system. Not surprisingly, social networks occupy the second position. In the third place, we can find cloud services belonging to Google and some CDN used for image storage. Overall, we can map URLs to categories easily. The system dramatically simplifies manual labeling, thanks to the rich information offered by URLs in each cluster.[8]

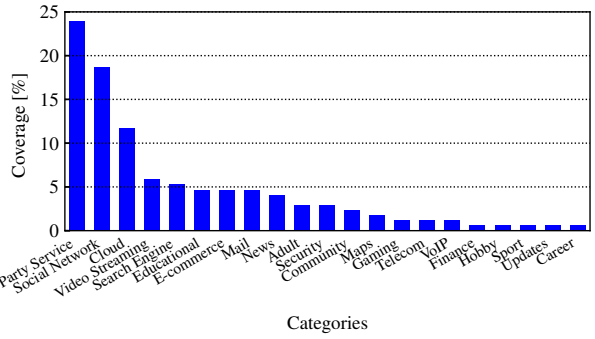



Figure 10: Most popular categories extracted from the clusters visited by at least two users.

## VII. System Engineering

In this section, we present our engineering effort to provide a scalable implementation of the LENTA system. The challenge is to complete the whole processing before the next period starts, *i.e.*, within $\Delta T$. Recall that each iteration of DBSCAN has a complexity of $O(N^2)$ [14]. It requires the computation of all $N^2$ distances for all pairs of URLs, which is based on the Levenshtein distance, whose complexity is $O(|s_1|, |s_2|)$, being N the number of URLs and $s_i$ the length of the $i^{th}$ URL. Notice that density-based clustering algorithms like DBSCAN need to access the pairwise distance between all nodes, and are, as such, intrinsically centralized. Our design focuses on the parallelization of computation of the distance matrix while keeping the clustering step centralized. For this, we take advantage of the Apache Spark environment, version 2.3.0, which runs on a medium-sized Hadoop cluster composed of 25 worker nodes with a total of 564 cores and 2TB of RAM. All the algorithms that need a centralized execution run on a high-end server equipped with two Intel® Xeon® E5-2640 processors providing 40 cores in total and 128 GB of RAM. All the code used in the experiments is written in Python, to guarantee consistency and uniformity.

### A. Distance Matrix Computation

LENTA is built on the top of DBSCAN and uses the silhouette to check cluster quality. Both require to access the distance matrix, *i.e.*, the matrix containing all the pairwise distances among all URLs.

To mitigate these limitations, we investigate parallelization techniques [13] considering both vertical and horizontal solutions. For vertical solution, we consider the standard centralized solution offered by Scikit-Learn function `pairwise_distances` which computes the distance matrix dividing the workload on multiple threads, each running on a separate CPU core on the same node. For horizontal scalability, we consider our implementation that leverages Apache Spark, where we distribute the computation of each distance pair among executors, which run on different nodes.[9]

We run a set of experiments to gather the execution time spent in completing the distance matrix calculation and the subsequent IDBSCAN execution afterward. We consider different datasets of an increasing number of URLs. We compare the two approaches in Fig. 11, with, respectively, 40 threads and 500 executors. Notice the log scale on the y-axis. The results depict the gain obtained by using the distributed approach (blue, dashed line) instead of the centralized one (red, solid line). For large datasets, we obtain a speed of about nine times. The distributed solution is, however, impacted by the initial startup time of the Spark job due to operations needed to initialize the environment and allocate the executors. This phase slows down the execution for small datasets. Figure 12 illustrates this behavior, reporting the speedup factor obtained dividing the execution time of the centralized implementation by the distributed one, considering a different number of executors. The red area highlights the limits for small datasets. For datasets with more than 10,000 URLs, the distributed implementation starts offering gains, scaling almost linearly with the number of executors. This analysis shows how it is possible to use a distributed approach for our problem, especially when the system has to deal with a considerable amount of data, taking advantage of the flexibility offered by horizontally scalable solutions. We, however, showed how, especially for smaller datasets, a centralized approach with a well/equipped machine is still preferable.

### B. System Knowledge Implementation

The second CPU intensive operation LENTA has to face is the comparison of each newly found cluster $\hat{C}(i)$ with those present in the System Knowledge $\hat{Z}(i)$. Also, this involves the computation of edit distances between many URL strings. For this, we develop the System Knowledge module in the Spark environment to guarantee scalability, both in terms of storage and performance. The idea again is to first compare in

[8]To let the reader verify this, we made the results publicly available at https://smartdata.polito.it/lenta-dataset

[9]The code is public and available at: https://github.com/marty90/spark-distance-matrix

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2927409, IEEE Transactions on Network and Service Management



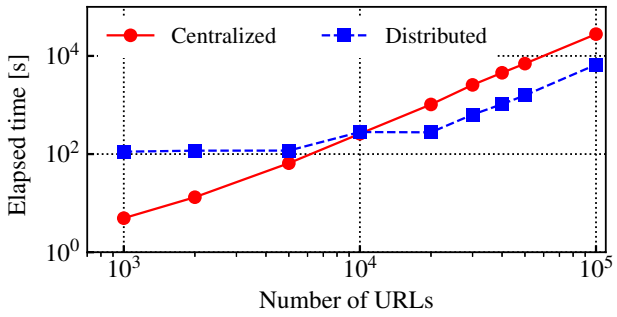



Figure 11: Overall time needed to execute the whole clustering for different dataset sizes.

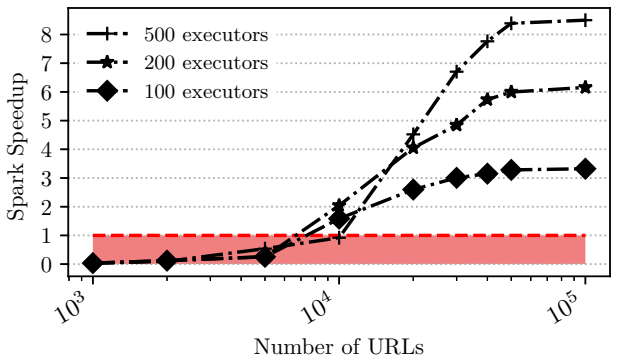



Figure 12: Speedup factor of Spark distributed approach varying the number of executors w.r.t. centralized algorithm with 40 threads.

parallel the clusters $\hat{C}_j \in \hat{\mathcal{C}}(t)$ with $\hat{Z}_j \in \hat{\mathcal{Z}}(t-1)$, $\forall j, l$ . In a second step, perform the insert and update operations on the System Knowledge representatives. Using a tree organization - through the implementation of the VP-tree [46] - for the representatives in $\hat{\mathcal{Z}}$, we improve in the look-up of the most similar cluster in the System Knowledge (eq. (8)). The VP-Tree structure is replicated in each executor and used to compute the k-NN (with $k = 1$) for each representative of each cluster $C_i(t)$. In the end, the algorithm updates $\hat{\mathcal{Z}}$, and build and redistributes the newly updated VP-tree. This technique speeds up the process and improves its scalability.

Computing and maintaining the System Knowledge for the two weeks of HTTPS traffic requires 34 minutes, while, for the three weeks of HTTP data, the whole process takes less than five hours.

## VIII. Conclusions

We designed a recursive version of a clustering algorithm over daily HTTP/HTTPS traffic generated by hosts in a network. We showed how LENTA reduces the amount of traffic that needs a manual check and eases the observation of changes in the network traffic. LENTA exposes well-formed clusters of URLs which significantly simplifies the identification of possibly malicious and undesired traffic.

This work goes in the direction of reducing the complexity of network traffic inspection, supporting the analysts in analyzing a few hundreds of clusters instead of several hundred of thousands of URLs. In general, LENTA simplifies the analysis of the traffic transported by a network. Our results show that the methodology, applied in a long-term observation, can identify anomalies in the traffic and changes in users' behavior.

## References

[1] Aggarwal, C.C., Reddy, C.K.: Data Clustering: Algorithms and Applications. Chapman and Hall/CRC (2013)
[2] Anderson, B.: Hiding in plain sight: Malware's use of tls and encryption. https://blogs.cisco.com/security/malwares-use-of-tls-and-encryption
[3] 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
[4] 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)
[5] 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
[6] 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
[7] Bilge, L., Kirda, E., Kruegel, C., Balduzzi, M.: Exposure: Finding malicious domains using passive dns analysis. In: Ndss. (2011)
[8] 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
[9] 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
[10] Cordova, I., Moh, T.S.: Dbscan on resilient distributed datasets. In: High Performance Computing & Simulation (HPCS), 2015 International Conference on, IEEE (2015) 531–540
[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] Faroughi, A., Javidan, R.: Canf: Clustering and anomaly detection method using nearest and farthest neighbor. Future Generation Computer Systems **89** (2018) 166 – 177
[13] Faroughi, A., Javidan, R., Mellia, M., Morichetta, A., Soro, F., Trevisan, M.: Achieving horizontal scalability in density-based clustering for urls. In: 2018 IEEE International Conference on Big Data (Big Data), IEEE (2018) 3841–3846
[14] Gan, J., Tao, Y.: Dbscan revisited: mis-claim, un-fixability, and approximation. In: Proceedings of the 2015 ACM SIGMOD International Conf. on Management of Data, ACM (2015) 519–530
[15] Gao, H., Hu, J., Wilson, C., Li, Z., Chen, Y., Zhao, B.Y.: Detecting and characterizing social spam campaigns. In: ACM IMC. (2010)
[16] Giordano, D., Traverso, S., Grimaudo, L., Mellia, M., Baralis, E., Tongaonkar, A., Saha, S.: Youlighter: A cognitive approach to unveil youtube cdn and changes. IEEE Transactions on Cognitive Communications and Networking **1**(2) (2015) 161–174
[17] 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
[18] Gu, G., Perdisci, R., Zhang, J., Lee, W.: Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. (2008)
[19] Gu, G., Zhang, J., Lee, W.: Botsniffer: Detecting botnet command and control channels in network traffic. (2008)
[20] Han, D., Agrawal, A., Liao, W.K., Choudhary, A.: A novel scalable dbscan algorithm with spark. In: Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International, IEEE (2016) 1393–1402
[21] Hartigan, J.A.: Clustering Algorithms. Wiley, New York (1975)

[22] Huang, F., Zhu, Q., Zhou, J., Tao, J., Zhou, X., Jin, D., Tan, X., Wang, L.: Research on the parallelization of the dbscan clustering algorithm for spatial data mining based on the spark platform. Remote Sensing **9**(12) (2017) 1301

[23] 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

[24] 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

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

[26] 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

[27] 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

[28] Lulli, A., Dell'Amico, M., Michiardi, P., Ricci, L.: Ng-dbscan: scalable density-based clustering for arbitrary data. Proceedings of the VLDB Endowment **10**(3) (2016) 157–168

[29] Luo, G., Luo, X., Gooch, T.F., Tian, L., Qin, K.: A parallel dbscan algorithm based on spark. In: Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom), 2016 IEEE International Conferences on, IEEE (2016) 548–553

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

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

[32] 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

[33] 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

[34] Patwary, M.A., Palsetia, D., Agrawal, A., Liao, W.k., Manne, F., Choudhary, A.: Scalable parallel optics data clustering using graph algorithmic techniques. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM (2013) 49

[35] 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

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

[37] Porras, P.A., Saïdi, H., Yegneswaran, V.: A foray into conficker's logic and rendezvous points. LEET **9** (2009) 7

[38] 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

[39] Stone-Gross, B., Cova, M., Cavallaro, L., Gilbert, B., Szydlowski, M., Kemmerer, R., Kruegel, C., Vigna, G.: Your botnet is my botnet: analysis of a botnet takeover. In: Proceedings of the 16th ACM conference on Computer and communications security, ACM (2009) 635–647

[44] Wang, B., Zhang, C., Song, L., Zhao, L., Dou, Y., Yu, Z.: Design and optimization of dbscan algorithm based on cuda. arXiv preprint arXiv:1506.02226 (2015)

[40] 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

[41] 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

[42] Trevisan, M., Giordano, D., Mellia, M., Munafó, M.: Five years at the edge: Watching internet from the isp network. In: 14th International Conference on emerging Networking EXperiments and Technologies (CoNEXT 2018). Volume 1. (2018)

[43] 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

[45] Wright, C.V., Monrose, F., Masson, G.M.: On inferring application protocol behaviors in encrypted network traffic. Journal of Machine Learning Research **7**(Dec) (2006) 2745–2769

[46] Yianilos, P.N.: Data structures and algorithms for nearest neighbor search in general metric spaces. In: SODA. Volume 93. (1993) 311–321


## BIOGRAPHIES

**Andrea Morichetta** (S'17) In December 2015 he received the Master Degree in Computer Engineering, from Politecnico di Torino. He joined the Telecommunication Networks Group in March 2016 with a scolarship and, from November, as a PhD student, always under the supervision of Prof. Marco Mellia, with a grant fully funded by the BIG-DAMA project. In summer 2017 he had a summer internship at Cisco in San Jose, CA. His research interests are in the fields of traffic analysis, security and data analysis.

**Marco Mellia** (M'97-SM'08) graduated from the Politecnico di Torino with Ph.D. in Electronics and Telecommunications Engineering in 2001, where he held a position as Full Professor.

In 2002 he visited the Sprint Advanced Technology Laboratories, CA. In 2011, 2012, 2013 he collaborated with Narus Inc, CA, working on traffic monitoring and cyber-security system design. Since 2015 he is collaborating with Cisco Systems for the design of cloud monitoring platforms. His research interests are in traffic monitoring and analysis, and in applications of Big Data and machine learning techniques for traffic analysis, with applications to Cybersecurity and network monitoring in general. He has co-authored over 250 papers and holds 9 patents. He was awarded the IRTF Applied Networking Research Prize in 2013, and several best paper awards. He is Area Editor of ACM CCR, and part of the Editorial Board of IEEE/ACM Transactions on Networking. He was program chair of several conferences, including ACM CoNEXT'17, IEEE TMA'16 and ITC'15.