POLITECNICO DI TORINO
Repository ISTITUZIONALE

LENTA: Longitudinal Exploration for Network Traffic Analysis

Original
LENTA: Longitudinal Exploration for Network Traffic Analysis / Morichetta, Andrea; Mellia, Marco. - ELETTRONICO. - (2018), pp. 176-184. (Intervento presentato al convegno ITC 30 - 2018 tenutosi a Vienna, AU nel 3-7 September 2018) [10.1109/ITC30.2018.00035].

Availability:
This version is available at: 11583/2715459 since: 2018-10-20T11:38:08Z

Publisher:
IEEE

Published
DOI:10.1109/ITC30.2018.00035

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

12 October 2024

LENTA: Longitudinal Exploration for Network Traffic Analysis

Andrea Morichetta, Marco Mellia

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

***Abstract*—In this work, we present LENTA (Longitudinal Exploration for Network Traffic Analysis), a system that supports the network analysts to easily identify traffic generated by services and applications running on the web, being them benign or possibly malicious. First, LENTA simplifies analysts' job by letting them observe few hundreds of clusters instead of the original hundred thousands of single URLs. Second, it implements a self-learning methodology, where a semi-supervised approach lets the system grow 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 lets the analysts easily observe changes in the traffic, like the birth of new services, or unexpected activities.**

We follow a data driven approach, running LENTA on real data. Traffic is analyzed in batches of 24-hour worth of traffic. We show that LENTA allows the analyst to easily understand which services are running on their network, highlights malicious traffic and changes over time, greatly simplifying the view and understanding of the traffic.

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 [11]. Video streaming, music, VoIP, chat, and traditional access to web pages today run on the top of HTTP or HTTPS. Even malware prefers HTTP as protocol to, e.g., let infected clients communicate to command and control (C&C) servers [2]. 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 which services are running in the network. To give the intuition of the variety of traffic today, Fig. 1 reports the growth in the number of unique URLs that are observed in a real network where hundreds of users are connected to the Internet. Data refers to March 2016, where still more than 40% of traffic was carried by HTTP [7]. As it can be seen, every hour several tens of thousands unique URLs (solid curve - left y-axis) are accessed via HTTP, with the total number (dotted curve - right y-axis) that grows to more than unique 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 detected. They need to process a consistent amount of traffic so to guarantee the correlation and comparison between events that in a too detailed analysis would be missed. This work needs clearly the support of automatic tools to process, analyze and extract useful information from the raw data.

The research leading to these results has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-129, "BigDAMA"

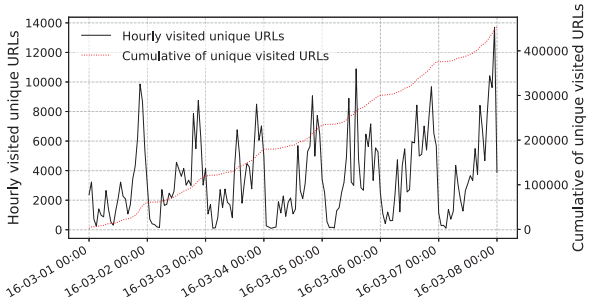


Figure 1: Evolution of unique URLs observed on a real network.

In this context, big data approaches are starting to emerge to scale the analysis of traces [3], [4], [6], [9], [13]. They offer the ability to process massive data [9], and run machine learning methodologies for traffic classification [6], [13], traffic monitoring analytics [4], or in general to support the so called data science process, i.e., the extraction of insights from massive data [3]. 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 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 applications fetching objects (e.g., objects in a web page, or system component for a web-app) [15], including malware that periodically contact C&C server or execute automatic actions. These latter have often a regular syntax, which makes them strictly different, but similar in the format. Designing a clustering solution of URLs requires ingenuity, given URLs are strings, for which the notion of similarity and distance is not trivial to define.

In this article, we propose LENTA (Longitudinal Exploration for Network Traffic Analysis). Here, first, we improve

978-0-9883045-5-0/18/$31.00 ©2018 ITC
DOI 10.1109/ITC30.2018.00035

classic clustering algorithms by automating the choice of parameters, an often cumbersome process. We demonstrate that this strategy offers better results with respect to what we obtained in our previous work CLUE (Clustering for URL Exploration) [10], using the original DBSCAN algorithm. Second and more important, we design a self-learning approach that lets the system build 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 known traffic is automatically labeled. This highlights changes and birth of previously unseen applications in the traffic pattern, building a longitudinal view of traffic.

We test LENTA on a real use case where a passive probe 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 LENTA ability in creating few clusters, which are easy to investigate and associate to services or malicious activities, and the capability of identifying new traffic generated by previously unknown systems. For instance, in our experiment LENTA lets us discover traffic related to well-known services (e.g., CDN, video services, online tracking and advertisement systems), unexpected applications (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.

II. Motivation and System overview

In this paper, we target the analysis of HTTP traffic, which still today amounts to more than 40% of web traffic, according to global statistics [7]. Furthermore, the majority of malicious traffic is in HTTP too [2], while well-behaved services are moving on HTTPS.

A. Motivation

We chose to 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, while URLs of different services should be grouped separately. We provide some examples 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 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) techniques. *B1* and *B2* illustrate two URLs generated by Sony connected Smart-TVs which access

Table I: Examples of similar URLs

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

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, there are services which 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.

B. System Overview

Fig.2 sketches the overall process. Our goal is to group all those URLs in the same cluster 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, collected URLs are clustered in $\mathcal{C}(i)$. Several challenges arise here, from the computation of the similarity between two URLs (i.e., strings), to the proper choice of clustering algorithm, from the parameter settings, to a scalable design.

Once clusters are identified, we reduce the dimensionality of the data by applying a sampling process, i.e., by extracting a summary of URLs found in each of them, 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 added to the System Knowledge after eventually an inspection of the network analyst to 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.

III. Methodology

A. URL Extraction

The first step of the process is to extract URLs from HTTP traffic. Visibility in HTTP traffic can be obtained using a

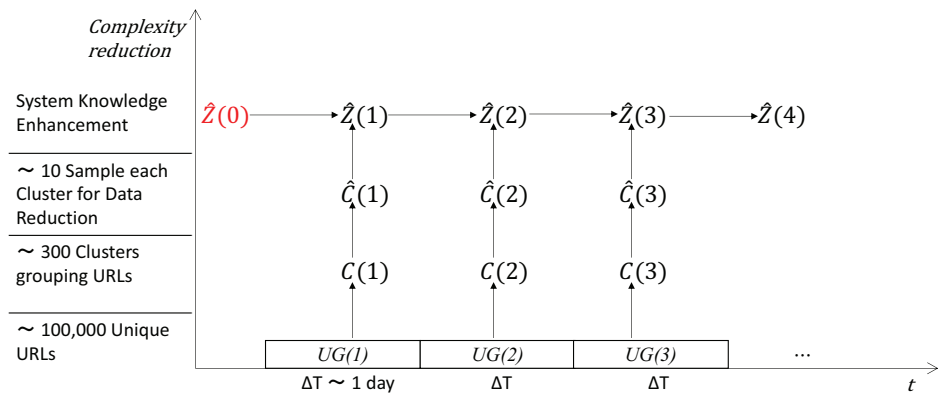


Figure 2: System overview

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 Tstat [14], a scalable passive network monitor 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 the traffic. For our experiment we use a one-week-long HTTP trace collected in March 2016 in an ISP network. To protect users privacy, all parameters in the URL have been removed, and only unique URLs were saved[1]. As shown in Fig. 1, we observe more than 430 000 unique URLs during a week, more than 60 000 per day (detailed in first row of Table IV).

Every period of duration ΔT, a URL group $UG(i)$ is formed and analyzed. In our experiments, we choose $\Delta T = 24h$. This is justified by the daily periodicity of traffic (see Fig. 1) which reflects the typical daily periodicity of users.

B. Distance definition

Clustering is the task of grouping a set of objects in such a way that the ones in the same group (i.e., the cluster) are more similar to each other than to those in other groups. We build on DBSCAN [1] to design a proper clustering algorithm.

In our case, objects are URLs, i.e., strings, for which there is no well-accepted notion of similarity. As such, we focus on a particular class of similarity metric, the edit-distance [5]. 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. We propose a custom modification of the Levenshtein distance, d_{LVS} [8]. Specifically, 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,

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

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 different.

C. Self-tuning Clustering

For clustering, we built upon and improve the well-known DBSCAN algorithm. DBSCAN falls under the family of the density based clustering techniques, where 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 center. A neighborhood is dense if there are at least MinPoints in the sphere of radiusϵ.

Despite the good results, the setting of the MinPoints and ϵ parameters remains open. In particular, MinPoints can be reasonably set using domain knowledge since it represent the minimum number of elements of a cluster. ϵ is instead hard to get, especially if the used distance is not well known. In the original CLUE, ϵ was manually selected. Here we

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

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 proper setting of ϵ, and each time accepting only those clusters that are well-shaped. Objects in bad-shaped clusters are eventually re-clustered in the next step, 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 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 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 $k =$ 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 $\eta = 75\%$ of points. We set $\epsilon = d_{th}$. With this choice, 75% of objects have at least $k =$ MinPoints objects at a distance smaller than 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 [12]. 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 i with all points in its cluster. Let $b(i)$ be the minimum among average distance of i to points in other clusters. In formulas, we have:

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

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

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

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 indicate that i might have been assigned to the wrong cluster. The average $S(C) = E[s(i), i \in C]$ over all points in cluster C is a measure of how tightly grouped all the elements in C are.

Given a cluster C, we say it is well-shaped if $S(C) > S_{min}$. Therefore, 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.

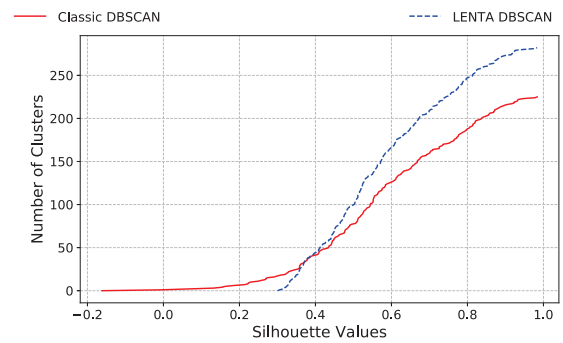


Figure 3: CDFs obtained by extracting the silhouette of clusters obtained with the classic DBSCAN and the updated algorithm proposed in LENTA.

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

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

We ran several experiments to check the quality of clustering for different values of S_{min} and η. In a nutshell, the algorithm is robust to the choice of η, 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.

The benefits of this self-tuning clustering are shown in Fig. 3. Here we report the silhouette values of clusters obtained running the classic DBSCAN algorithm and the self-tuning version over one day of traffic, with more than 59 000 URLs. Bad clusters (S_{min} < 0.3) are recomputed and separated in more meaningful groups, increasing both the cohesion and the number of final clusters, that in this experiment grows by 25% (from 226 clusters of the classic DBSCAN to 283).

D. Sampling for Data Reduction

Next, we sample a subset of elements from each cluster. The rationale is twofold: to ease the comparison between clusters reducing computational complexity, while maintaining their information quality; and to keep a digest of the collected traffic in the System Knowledge, reducing its footprint.

We sample each cluster $C_j \in \mathcal{C}$ using 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[2], we select elements as $\hat{C}_j = sample(C_j, m)$.

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

- Random sampling: selecting m objects at random from the elements of C_j, i.e., $sample(C_j, m) = rand(C_j, m)$;

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

- Percentile sampling: selecting the subset of 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 intra-cluster distances for each URL $s_i \in C_j$

$$\left\{E_{s_k \in C_j}[d_{URL}(s_i, s_k)], \forall s_i \in C_j\right\} \quad (1)$$

The elements selected 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 subsamples 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[3]. The medoid is generally a good choice to describe a group of elements, but it is more appropriate for spherical and homogeneous clusters. Being a cluster in DBSCAN 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 is more peculiar to the population of the cluster.

E. System Knowledge enhancement intuition

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

$$d_{min}(\hat{C}, \hat{\mathcal{Z}}) = \min_{\hat{Z} \in \hat{\mathcal{Z}}} \left(d\left(\hat{C}, \hat{Z}\right)\right)$$
$$\text{where } d(\hat{C}, \hat{Z}) = \min_{\substack{c \in \hat{C} \\ z \in \hat{Z}}} d_{URL}(c, z) \quad (2)$$

Let $\hat{\mathcal{C}}(t)$ 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 $\hat{Z}_l(t-1) \in \mathcal{Z}(t-1)$ is

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

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

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

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

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, that is associated to the old one, 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 use 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,

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

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

G. Implementation and complexity

LENTA has been implemented using the Apache Spark framework. Running DBSCAN over N elements requires the computation of $O(N^2)$ distances between each pair of elements. This results in a very expensive task, since d_{URL} complexity is $O(len(s_1), len(s_2))$ and URLs can be very long strings. Spark parallelism helps to compute the N^2 distance matrix by letting each executor compute a subset of the entire matrix. For instance, considering a data set of approximately 60 000 elements, a single executor requires more than 24 hours, while a Spark clusters with 60 active executors less than 1 hour. After computing the matrix, the iterative clustering can run on a single executor without penalties, while the System Knowledge enhancement step complexity is bounded by the sampling process. These last two steps are completed in less than 30 min on a single executor. More details are provided in Sec. IV-B.

IV. Results

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

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.

Top part of Tab. II shows the biggest clusters, while bottom part those with the highest silhouette. 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 from granted, as often the complexity of URLs structure tend to increment the distance also for actually similar elements.

y, include e-commerce websites, blog services, chat platforms, etc.

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 apparently random

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

Table II: 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

paths. After further analysis[4], 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 a specific content. For example the Chinese messaging system `msg.71.am` is divided into two clusters, one serving images (.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.

B. Parameter setting

Once clusters are identified, we extract a digest via sampling. This represent the most critical step, since is essential to balance representativeness, and the complexity of the System Knowledge enhancement step. 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 α 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 obtained from the first day $\mathcal{C}(0)$ into two sets. The first part 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. Sampling is applied, and $\hat{\mathcal{C}}(1)$ is compared to $\hat{\mathcal{Z}}(0)$. We expect half of the clusters to be identified as already known, and half to be new.

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

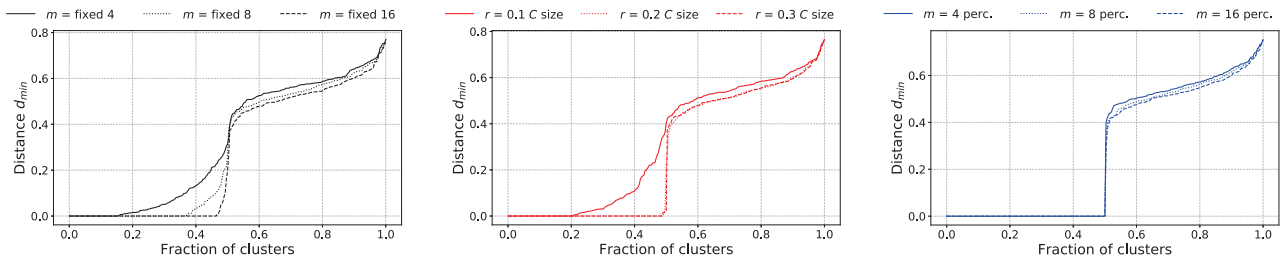
Results are depicted in the plots of Fig. 4 which show $d_{min}(\hat{C}_j(1), \hat{\mathcal{Z}}(0))$, in increasing order, respectively comparing results for fixed random $m = \{4, 8, 16\}$, $r = 0.1, 0.2, 0.3$ and finally for m fixed percentile based samples $m = \{4, 8, 16\}$.

We would expect to see an approximation of a step curve, where the first half of the distances are equal to 0 because the same clusters are compared; the second half of the distances have a value larger than 0 – the higher the better. Fig. 4 clearly shows that, in case of random sampling, the more the number of samples, 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%.

Situation improves when using percentiles, whose smart sampling guarantees best results. Indeed, when we consider the percentile, we always obtained a perfect distance of 0 for the clusters that contains the same elements of the compared ones. That is happening because two sets are equal and we deterministically select the points.

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 larger than the threshold α. Fig. 4a, Fig. 4b and Fig. 4c 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 $\alpha = 0.3$.

As a drawback, increasing the number of representatives increases the computation complexity, due to the need to compute $O(m^2)$ $d_{URL}(.)$. Fig. 5 shows the experimental computational time using lin/log scales. For variable fraction r, 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%



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

Figure 4: d_{min} when 50% of traffic is the same and 50% is new. Different choices of sampling approaches.

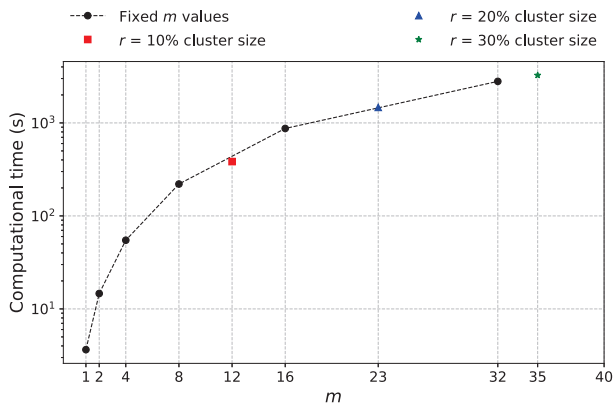


Figure 5: Computation time for different sampling strategies.

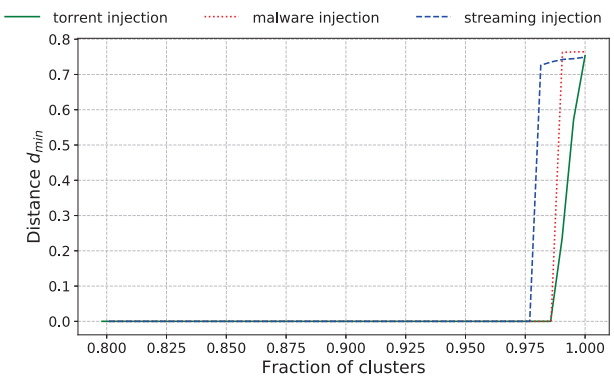


Figure 6: Curves of distances when new traffic is injected in the controlled experiment. Top 20% clusters are reported.

or 30% that already have a complexity 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 $m = 16$.[5]

V. Evolution over time

In this section we show the results of running LENTA on a real scenario. We first consider a controlled experiment and then we apply LENTA over 7 days of traffic collected from the ISP network.

A. *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., $UG(3) = UG(2) \cup \{StreamingURLs\}$.

[5]Also in this case CPU time can be reduced by computing d_{URL} in parallel.

After each stage, we run LENTA and check if it is able to identify the new traffic. Results are reported in Fig. 6, 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 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. III 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`.

Table III: New clusters highlighted during the comparison with the system knowledge.

Experiment stage	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

Table IV: 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

B. Real case scenario

We now run LENTA on a one week of data collected in an ISP network. Table IV detail results. Figure 7 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). 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 less than 170 clusters per day. Compare the growth with Fig 1, where the number of unique URLs tops to more than 420 000, with on average 72 000 unique URL 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 unique URLs.

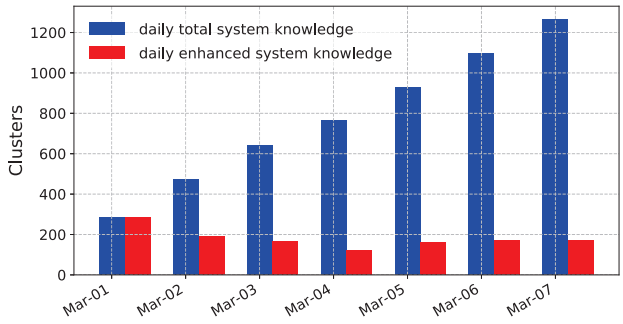


Figure 7: Daily enhancement of system knowledge

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 previous collected traffic, i.e., those for which $d_{min}(C_j(t), \hat{\mathcal{Z}}(t-1))$ is higher.

Tab. V details the results. Also in this case, 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 webportal `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 contain 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: an uncommon services like `aww.com.au`, an australian news website, and webpages translated using the Google Translate online service (curiously translating adult content website, possibly to evade content filtering policies).

VI. 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. We 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 to reduce the traffic to manually check and

Table V: 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**

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 of identifying anomalies in the traffic.

In this work we have focused our attention to HTTP traffic. The promising results suggest to focus on HTTPS traffic too, in order to have a complete view on the network activities. LENTA would indeed work with no changes, assumed visibility in HTTPS traffic if possible. For example, 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. Those techniques, together with proper privacy preserving practices, would extend the view to the full scenario.

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.

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] 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

[4] 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

[5] 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

[6] 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

[7] 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

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

[9] 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

[10] 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

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

[12] 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

[13] 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

[14] 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

[15] 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