YouLighter: An Unsupervised Methodology to Unveil YouTube CDN Changes

Danilo Giordano[1], Stefano Traverso[1], Luigi Grimaudo[1],
Marco Mellia[1], Elena Baralis[1], Alok Tongaonkar[2], Sabyasachi Saha[2]
[1]Politecnico di Torino - first.last@polito.it
[2]Symantec Corporation - {alok_tongaonkar,saby_saha}@symantec.com

***Abstract*—YouTube relies on a massively distributed Content Delivery Network (CDN) to stream the billions of videos in its catalogue. Unfortunately, very little information about the design of such CDN is available. This, combined with the pervasiveness of YouTube, poses a big challenge for Internet Service Providers (ISPs), which are compelled to optimize end-users' Quality of Experience (QoE) while having no control on the CDN decisions.**

This paper presents *YouLighter*, an unsupervised technique to identify changes in the YouTube CDN. *YouLighter* leverages only passive measurements to cluster co-located identical caches into *edge-nodes*. This automatically unveils the structure of YouTube's CDN. Further, we propose a new metric, called *Pattern Dissimilarity*, that compares the clustering obtained from two different time snapshots, to pinpoint sudden changes. While several approaches allows us to compare the clustering results from the *same* dataset, no technique measures the similarity of clusters from *different* datasets. Hence, we develop a novel methodology, based on the *Pattern Dissimilarity*, to solve this problem.

By running *YouLighter* over 10-month long traces obtained from ISPs, we pinpoint both sudden changes in edge-node allocation, and modifications to the cache allocation policy which actually impair the QoE that the end-users perceive.

I. Introduction

YouTube is one of the most popular and demanding Internet services. It accounts for 1 billion users distributed world-wide, who watch 6 billion hours of videos per month.[1] Due to its popularity and the nature of the content that it distributes, the demanded load to handle is huge, and guaranteeing a satisfactory Quality of Experience (QoE) for the users is a challenging task to accomplish. To this end, YouTube leverages a massive, globally distributed Content Delivery Network (CDN), the Google CDN [1], which consists of hundreds of *edge-nodes* scattered in the Internet. Each edge-node hosts hundreds of video servers, or *caches*, which can each potentially serve any video a user may request [2].

Google, as many other Over-the-Top content providers, places its edge-nodes close to users, usually at aggregation points directly peering with Internet Service Providers (ISPs).[2] Hence, Google uses the ISP's network as the "last mile" to deliver YouTube videos. Despite this localized setup, once a user requests a video playback, the CDN load balancing algorithm directs the request to one of the caches, and there is no mean to influence or predict which cache, or even which edge-node will be used [3], [4]. This is particularly critical for the ISP, which on the one hand is compelled to deliver YouTube videos to users without impairing the QoE, while on the other aims at minimizing the delivery costs. Hence, the ISP spends a significant effort in monitoring the CDN infrastructure and designing ad hoc traffic engineering policies for YouTube traffic [5]. However, changes in the YouTube CDN occur frequently, and they may involve modifications in the infrastructure, e.g., the activation of a new cache, or in the load balancing algorithm decision, e.g., a sudden switch of caches to serve requests. Conversely, the ISP's policies are often static and hardly cope with the continuous evolution of the Google CDN: any sudden change can make the ISP's optimization obsolete, and thus ineffective, possibly causing abrupt disruptions or QoE degradations. This constitutes an issue for the ISP, as it sees its reputation degrade when a change happens, even if Google caused it.

In this paper, we present *YouLighter*, a novel methodology to automatically monitor and pinpoint changes in the YouTube CDN. *YouLighter* relies on an unsupervised learning approach that, as such, does not require any knowledge of the YouTube infrastructure. Instead, it only assumes that the ISP has deployed passive probes, which expose TCP flow level logs summarizing video requests from users. Considering a given observation window of, say one day, *YouLighter* aggregates these flow logs to constitute a *snapshot* of the traffic exchanged with YouTube caches. Based on DBSCAN [6], a well-established unsupervised machine learning algorithm, *YouLighter* is able to automatically group thousands of caches into a bunch of edge-nodes using simple features that characterize the network distance of caches from the vantage point.

To pinpoint changes, *YouLighter* runs DBSCAN on consecutive snapshots, it transforms the corresponding clustering results into *centroid patterns*, and it compares them using the notion of *Pattern Dissimilarity*. The bigger the distance between two snapshots is, the more different the sets of YouTube caches to serve ISP customers during the two periods of time are. *YouLighter* highlights several kinds of changes. E.g., deviations from the typical behavior of edge-nodes possibly induced by congestion arising in the network. In general, *YouLighter* unveils sudden changes happening the YouTube CDN infrastructure which may be responsible of QoE issues for ISP customers.

We validate our methodology over traces we collect from four different vantage points that we have deployed in two ISPs in two different countries. First, we demonstrate that the clustering algorithm *YouLighter* adopts is effective at identifying and grouping YouTube caches belonging to different edge-nodes. Second, we run *YouLighter* over different collected snapshots considering the longitudinal dataset, which, overall,

[1]https://www.youtube.com/yt/press/statistics.html
[2]https://peering.google.com/about/index.html

accounts for more than 33 months of traffic. We pinpoint several examples of sudden and previously undiscovered changes in the YouTube CDN. For some of them, we investigate the impact on the QoE of ISP customers, revealing the sudden drop of average video download throughput to less than 250 kb/s which hampers even the possibility of watching a video.

We believe that *YouLighter* is a promising tool for ISPs, network administrators and researchers to monitor the YouTube CDN and the traffic it generates. Importantly, thanks to its design, *YouLighter* offers the capability of automating and accelerating the troubleshooting procedures. For instance, ISPs may use *YouLighter* to quickly react to changes possibly harming customers' QoE. In this direction, ISPs may adopt traffic engineering algorithms to change routing to underperforming edge-nodes, e.g., by means of BGP policies, or to enforce DNS policies to overrule YouTube choices by redirecting traffic from caches with bad QoE to caches with a good QoE.

Finally, while we engineer *YouLighter* to target YouTube CDN monitoring, we believe that the *Pattern Dissimilarity* notion we introduce in this paper constitutes a more general framework that has the potential to open the usage of unsupervised algorithms for anomaly detection problems in general.

The remainder of this paper is structured as follows: Sec. II discusses the related work. Sec. III describes the details of our datasets, and shows the dynamicity of YouTube cache selection policies. Sec. IV presents our methodology and introduces the *Pattern Dissimilarity*. Sec. V presents our results: First, we evaluate the sensitivity of *YouLighter*'s parameters, and, second, we show how effective *YouLighter* is at pinpointing changes in YouTube CDN employing our traces. Finally, Sec. VII concludes the paper.

II. Related Work

A large body of work has analyzed the YouTube delivery infrastructure and its evolution over time [1], [2], [3], [7], [5]. They show a highly dynamic system which keeps changing over time due to continuous upgrades in the infrastructure [1], [2] or due to the dynamicity of the cache selection policies [3], [7]. Some of the findings are already outdated. For instance, the load-balancing policy based on HTTP redirections which is described in [3], [7] is no longer in place, and YouTube dismissed the naming scheme described in [2] at the end of 2011. To the best of our knowledge, the only updated work which resembles in spirit our study is [5]. However, the proposed methodology requires a significant manual effort, and the paper mostly focuses on results about the characterization of the YouTube service in terms of traffic characteristics and QoE perceived by the users. In this work, we do not aim to offer an updated view or characterization of YouTube. Instead, we present a methodology that allows to automatically identify changes in both the infrastructure, e.g., the appearance of new edge-nodes, and in the day to day management of the infrastructure, e.g., a change in the load-balancing algorithm that may affect millions of customers.

Our contribution is in line with the body of works focusing on anomaly detection, for which [8], [9] offer good surveys. To the best of our knowledge, only [10] targets large scale anomaly detection in operational networks. However, it presents a supervised system, which relies on data from passive probes, topology information and routing tables to feed a classic forecasting system, which finally compares its predictions to the actual measurements to pinpoint deviations. *YouLighter*, on the other hand does not assume any knowledge of a baseline, and leverages unsupervised algorithms to automatically unveil changes. We specifically design it to target the YouTube CDN, for which the ground truth is a moving target that is very difficult to know.

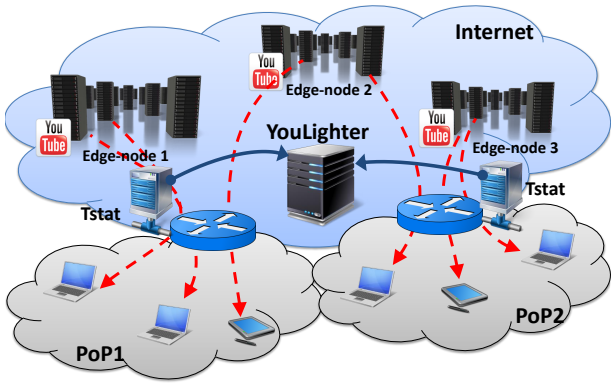


Fig. 1. The traffic monitoring setup we employ for this paper.

However, the application of unsupervised learning techniques to get insights about the network traffic is not new. For instance, [11] proposes a flow-based anomaly detection algorithm based on k-means, while [12] uses DBSCAN to identify anomalous clusters. In all the cases, clustering is used to study the same given dataset. To the best of our knowledge, no approaches have been proposed to identify anomalies by comparing clustering results attained from different datasets (e.g., different time snapshots, different populations, etc.). Only [13] aims at measuring similarity between sets of overlapping clusters from complex networks, in which groups of nodes form tightly connected units linked to each other. Since points are not embedded in a metric space, they define ad-hoc distances. *YouLighter* operates in a geometric space where we can exploit the concepts of density and centroid of a cluster to simplify the comparison among two different datasets.

YouLighter differs also from techniques for the tracking of moving clusters and objects as [14], [15]. Indeed, their goal is to track the movements of the same clustered objects over time, e.g., a group of migrating animals. On the contrary, *YouLighter* has no insights about the CDN infrastructure and it cannot track single objects, which may disappear and reappear freely.

Finally, other approaches as [16] measure the similarity among sample distributions obtained at different time intervals. However, directly relying on distributions to perform the comparison considerably complicates the detection of the edge-nodes behind the changes. Instead, *YouLighter* extracts and compares clustering patterns, which are simpler to process in an automatic manner, and allow to immediately pinpoint the edge-nodes (i.e., the clusters) responsible for possible deviations.

III. Datasets

We assume the ISP has instrumented the network with passive probes, which collect statistics from flows carrying YouTube videos. In this work, we rely on passive probes running Tstat [3] that we install in Points-of-Presences (PoPs) of

[3] http://tstat.polito.it

Trace	Period	Volume	Flows	Caches
ISP1-A	01/04/2013 - 28/02/2014	138.7 TB	33,216,794	8,664
ISP1-B	01/04/2013 - 28/02/2014	152.9 TB	31,643,603	8,899
ISP1-C	01/04/2013 - 28/02/2014	134.8 TB	27,377,089	9,028
ISP2	01/03/2014 - 17/07/2014	48.3 TB	9,100,163	3,755

TABLE I. TRACES CONSIDERED IN THIS STUDY.

operational networks, as depicted in Fig.1. Clients are located in one PoP, and connect to the backbone via a router, where Tstat monitors the traffic. Tstat observes packets, rebuilds each TCP flow, tracks it, and at the end of flow, logs detailed statistics. Tstat can classify TCP flows that carry YouTube videos. For each request, it logs i) the anonymized client IP address, ii) the server IP address, iii) the hostname of the server, iv) the TCP minimum Round Trip Time (RTT),[4] v) the IP Time-To-Live (TTL) of packets received by the client in the PoP, vi) the amount of bytes the clients send and receive, vii) the average download throughput, and viii) the time at which the TCP connection starts. Note that Tstat computes all these metrics considering only TCP segments, and do not require access to application payload. This avoids any privacy issues. Moreover, this allows us to collect all needed statistics even in presence of encryption, e.g, HTTPS which nowadays represents more than 50% of overall YouTube traffic [17].

We have been collecting traffic logs since April 2013 by monitoring the traffic users generate when accessing the Internet. We instrument four different probes. Three of them are located in PoPs of the same ISP and in two different cities of the same country. We install the fourth one in a PoP of a different ISP in a second country. Tab. I describes, for each trace, the time period, the total downloaded volume, the number of flows and the number of YouTube servers we observe. Notice that in total we monitor the activity of more than 32,000 customers, and the maximum number of caches that ISP1 customers used at least once is ∼9,000.

A. YouTube Cache Naming Structure

We find that the YouTube infrastructure described in [2] is no longer in use. Since 2012, YouTube server hostnames are in the form `rx---ABCxxtxx.c.youtube.com`, where `x` are numbers, while `ABC` is a three-letter code reporting the IATA code of the closest airport. For instance `r7---fra07t16.c.youtube.com` identifies a single cache, in Frankfurt. The hostname resolves to a single IP address, 74.125.218.182 in the example. Thus, we can uniquely identify a cache by its hostname.[5] All caches co-located in the same edge-node share the same IATA code. This allows us to get coarse ground truth about the location of servers. However, as we will see, several edge-nodes can be located in apparently different areas, but share the same IATA code.

We run some active experiments to cross-check if YouTube specializes caches to serve some particular content, and we verify that every cache can serve any video, at any resolution, in any format, e.g., MPG4 or Flash, to any device, e.g., PC, smartphones or tablets.

[4]The RTT is measured as the time between the client data segment and the corresponding server acknowledgment observed at the vantage point. For each TCP connection, minimum RTT is computed among all valid samples.

[5]Starting from January 2013, YouTube obfuscates the IATA code using a simple substitution cipher that we were able to break. For instance, `r7---fra07t16.c.youtube.com` becomes `r7---sn-4g57kued.c.youtube.com`. From October 2013, the `youtube.com` domain has been replaced by the `googlevideo.com` domain. This information can be used to identify YouTube flows even in presence of [18].

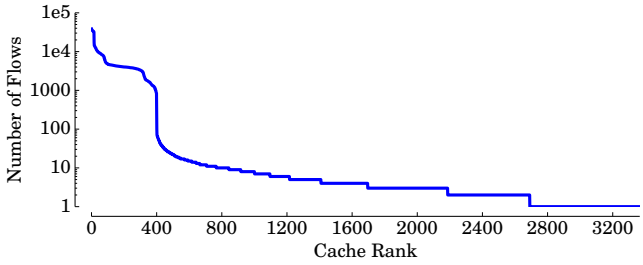


Fig. 2. Rank of YouTube caches based on the number of flows. February 2014, *ISP1-A*.

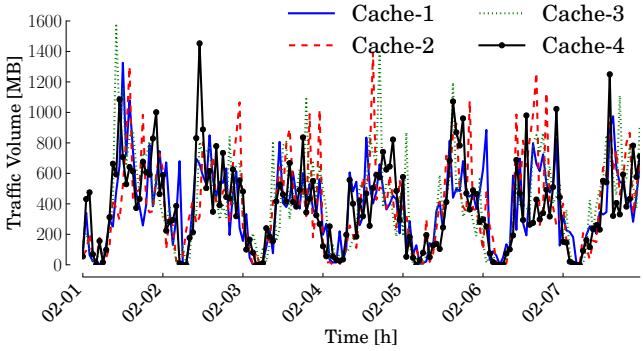


Fig. 3. Evolution of the volume of traffic for the four most active caches we observe on February 1st 2014. First week of February 2014, dataset *ISP1-A*.

B. Characterization of the Load Balancing Policies

Every time a user starts a video playback, the player starts a progressive download of the video content from the specific cache the system provides in the HTML page.[6] We are interested in seeing which are the policies governing the server allocation, such as (i) is there any "preferred" group of caches? or (ii) are those stable over time? Fig. 2 reports the rank of YouTube caches based on the number of flows they handle, which is proportional to the number of served videos. We consider February 2014 from the *ISP1-A* dataset. First, notice that we observe more than 3,200 caches during one month. Second, the load each cache handles is very heterogeneous; few servers handle lots of requests, but there is a not negligible number of caches that serves a significant portion of flows. For instance, more than 400 caches serve more than 100 videos, and in order to observe 95% of requests, one should monitor about 330 caches.

We also notice that the rank is extremely dynamic over time. For instance, we pick the four most active caches during the 1st of February 2014 and we report in Fig. 3 the amount of traffic they generate over time for the following seven days. As shown, the amount of traffic a single cache handles changes widely over time, and none of the monitored caches keeps a constant leading position for a long period of time. As one may expect this dynamicity to disappear when reducing the focus, we also run an experiment to monitor a larger pool of caches as those in the rank in Fig. 2, and we recompute the same rank on a daily basis for one month. We observe that

[6]Load balancing policies are implemented at application layer. Indeed the web server chooses and encodes the cache hostname directly in the HTML page served to the client.

the rank changes widely day by day and the most used cache in a day is not among the top-10 cache of the month (due to the lack of space we can not report here the full description of this experiment; more details are available in our technical report [19]). This shows that the server allocation policies adopted by YouTube spread the load over several hundreds of caches, and the choices are extremely dynamic over time if we observe with the fine grained granularity of a single cache.

Since caches inside the same edge-node are all equivalent, the intuition is to observe the system using the coarse granularity offered by edge-nodes. However, edge-nodes are unknown, they can change over time due to system upgrade or redesign, and information that could be available (e.g., the IATA code) may be not reliable, or may be removed by YouTube. In the following, we design an unsupervised clustering algorithm to automatically identify edge-nodes from just the observation of traffic flows.

IV. Methodology

Intuitively, the path between two caches in the same edge-node and clients in the same PoP exhibits the same properties, e.g., same RTT. Conversely, the path between two caches in different edge-nodes should present different RTT. This intuition is corroborated in Fig. 4 which depicts the 5th, 20th, 50th, 80th, and 95th percentiles of the per-cache RTT distribution. We identify caches with their IP address, and then we order and group them into edge-nodes using the IATA code as ground truth so that caches belonging to the same edge-node appear one close the other. Five edge-node are present, E-1 to E-5. Each hosts a variable number of caches, with E-3 being the largest. As shown, the caches in the same edge-node exhibits very similar RTT percentiles, suggesting that we can identify clusters of caches by considering the RTT as a feature.

A. Multi-dimensional Clustering

We leverage above intuition to design a clustering algorithm to automatically find homogeneous groups of caches. We use some ingenuity to characterize the path from client to each cache, and then to cluster caches that exhibits similar paths. We can split the process of our methodology into the following steps:

Step 1 - Passive monitoring of YouTube video flows: As described in Sec. III, a passive probe provides the continuous collection of YouTube traffic logs. We log each metadata of each TCP connection, and we store logs in a database for further processing.

Step 2 - Measurement consolidation and filtering: To ease the monitoring procedure, we use a batch processing approach that considers time windows of size ΔT. Thus, every ΔT we generate a "snapshot", and we aggregate and process measurements in it. In the following, we indicate the n-th snapshot as a superscript when needed, e.g., $a^{(n)}$ indicates the metric a at snapshot n.

We identify each cache x by its IP address. We then group all flows in the same snapshot with the same server IP address to obtain a table where columns correspond to the metric (e.g., RTT, TTL, transmitted packets, etc.), and each row corresponds to a sample, i.e., the tuple of measured values observed within a TCP flow. Since we are interested in the active caches, we discard those with less than $MinFlow = 50$ samples. We define the whole measurement snapshot n as $X^{(n)}$.

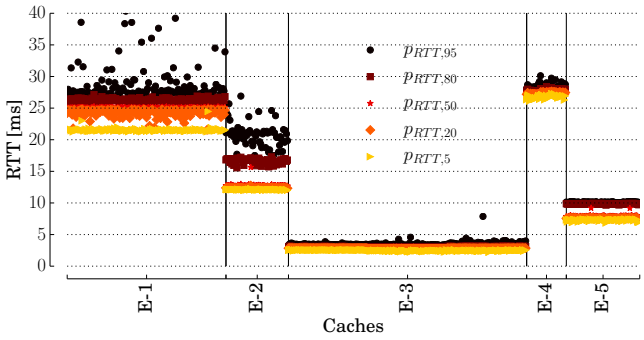


Fig. 4. Example of per cache RTT percentiles. Caches sorted by IP address, and grouped by (anonymized) IATA code.

Step 3 - Feature selection and data normalization: Next, we apply a feature selection driven by domain knowledge to select the set $\mathcal{M}$ of *metrics*. In particular, as we are interested in grouping caches according to the path properties, we choose $\mathcal{M} = \{RTT, TTL\}$. Then, for each cache x in the snapshot X, and for each metric $m \in \mathcal{M}$, we generate an empirical distribution. From the distribution, we extract the vector $P_m(x) = (p_{m,1}(x), p_{m,2}(x), \ldots, p_{m,k}(x))$ containing k percentiles of m for cache x. We thus standardize percentiles following a simple normalization:

$$min_m = \min\left(p_{m,i}(x)\ \forall x \in X, \forall i = 1, \ldots, k\right) \quad (1)$$

$$max_m = \max\left(p_{m,i}(x)\ \forall x \in X, \forall i = 1, \ldots, k\right) \quad (2)$$

$$\bar{p}_{m,i}(x) = \frac{p_{m,i}(x) - min_m}{max_m - min_m} \quad (3)$$

Intuitively, Eq.(3) normalizes the percentiles of metric m so that $\bar{p}_{m,i} \in [0, 1]$.

At last, $\bar{P}_m(x) = (\bar{p}_{m,1}(x), \bar{p}_{m,2}(x), \ldots, \bar{p}_{m,k}(x))$ represents the standardized vector of *features* for the metric m for server x. Recalling that $\mathcal{M} = \{RTT, TTL\}$, we identify each cache $x \in X$ with a $2k$-dimensional space of edge 1 by features:

$$\bar{x} = (\bar{P}_{RTT}(x), \bar{P}_{TTL}(x)) \quad (4)$$

and we transform the original set of caches X into a set of points $\bar{X} = \{\bar{x}\}$.

Step 4 - Clustering: We employ the density-based DBSCAN algorithm [6] to group together servers based on their multi-dimensional features. We choose DBSCAN because (i) it is able to handle clusters of arbitrary shapes and sizes; (ii) it is relatively resistant to noise and outliers; and (iii) it does not require the specification of the number of desired clusters. DBSCAN requires two parameters: ϵ and $MinPts$. ϵ determines the maximum allowed distance between any given point in a cluster and its closest neighbor belonging to the same cluster, and $MinPts$ the minimum number of points required to form a cluster. Based on that, it classifies all points as being (i) core points, i.e., in the interior of a dense region; (ii) border points, i.e., on the edge of a dense region; or (iii) noise points, i.e., in a sparsely occupied region. Noise points do not form any cluster, while the algorithm puts in the same cluster any two core points that are within ϵ of each other. Similarly, any border point that is close enough to a core point is put in the same cluster as the core point. The result of this process is a collection $\mathcal{C}$ of clusters $C_j \in \mathcal{C}$, also named as *clustering*:

$$\mathcal{C} = \{C_j\} = \text{DBSCAN}(\bar{X}) \quad (5)$$

B. Highlighting Changes with the Pattern Dissimilarity

We are now interested in tracking the evolution of clusters over time, for which, as we discuss in Sec. II, no known solution is present in the literature. Indeed, it is not obvious how to compare two clusterings $\mathcal{C}1$ and $\mathcal{C}2$ obtained considering two *different* datasets, i.e., snapshots in our case. For instance, i) points that were present in $\mathcal{C}1$ may not be present in $\mathcal{C}2$, and vice versa; ii) points clustered into the same cluster in $\mathcal{C}1$ can now belong to two or more clusters in $\mathcal{C}2$; and iii) the same points that form a cluster in $\mathcal{C}1$ can still form the same cluster, but can be placed in another region in the clustering space in $\mathcal{C}2$. In our case, this corresponds to i) popular caches at snapshot n that are not anymore used at snapshot $n+1$, or ii) some caches at snapshot n that were part of the noise are instead clustered at snapshot $n+1$, or iii) the path to caches suddenly changes at snapshot $n+1$, altering RTT and TTL.

To evaluate the difference among the clustering, we propose a novel methodology that is based on the notion of *Pattern Dissimilarity*.

1) Clustering Patterns: We first map each cluster into a single *Centroid* that summarizes it. Given a cluster $C \in \mathcal{C}$, we consider the centroid, or geometric center, $\hat{x}$ whose components $\hat{p}_{m,i}$ in the i percentile of feature m are:

$$\hat{p}_{m,i} = \frac{1}{|C|} \sum_{x \in C} renorm(p_{m,i}(x)) \tag{6}$$

All centroids then form a *pattern* $\hat{\mathcal{P}} = \{\hat{x}\}$. The $renorm()$ function eventually considers the re-normalization of features that can be needed if points in $\mathcal{C}1$ and $\mathcal{C}2$ went through different standardization processes. In our case, assuming $\mathcal{C}1 = \mathcal{C}^{(n)}$, $\mathcal{C}2 = \mathcal{C}^{(n+1)}$, from Eq.(3) for each $m \in \mathcal{M}$ we have:

$$Min_m = \min\left(min_m^{(n)}, min_m^{(n+1)}\right) \tag{7}$$

$$Max_m = \max\left(max_m^{(n)}, max_m^{(n+1)}\right) \tag{8}$$

$$renorm_m(a) = \frac{a - Min_m}{Max_m - Min_m} \tag{9}$$

2) Centroid Distance: Given a centroid $\hat{x}$ and a centroid pattern $\hat{\mathcal{P}}$, we define the *Centroid Distance* (CD) as the distance between $\hat{x}$ and its closest centroid in $\hat{\mathcal{P}}$. Specifically, we compute the closest centroid $\hat{y}^* \in \hat{\mathcal{P}}$ such that $d(\hat{x}, \hat{y}^*) \leq d(\hat{x}, \hat{y})\ \forall \hat{y} \in \hat{\mathcal{P}}$. $d(x, y)$ can be any distance metric that is valid in the feature space. In this work, we use the classic Euclidean distance. Thus, the Centroid Distance CD of the centroid $\hat{x}$ from centroids in $\hat{\mathcal{P}}$ is

$$CD(\hat{x}, \hat{\mathcal{P}}) = \min_{\hat{y} \in \hat{\mathcal{P}}} d(\hat{x}, \hat{y}) \tag{10}$$

Hence, the Centroid Distance couples centroids according to a nearest neighbor principle.

3) Pattern Dissimilarity: At last, we define the *Pattern Dissimilarity* - PD - as the sum of the Centroid Distance among every centroid in the clusterings. Since the number of clusters in $\hat{\mathcal{P}}1$ and $\hat{\mathcal{P}}2$ may be different, we need to symmetrize the definition:

$$PD(\hat{\mathcal{P}}1, \hat{\mathcal{P}}2) = \sum_{\hat{x} \in \hat{\mathcal{P}}1} CD(\hat{x}, \hat{\mathcal{P}}2) \ + \ \sum_{\hat{x} \in \hat{\mathcal{P}}2} CD(\hat{x}, \hat{\mathcal{P}}1) \tag{11}$$

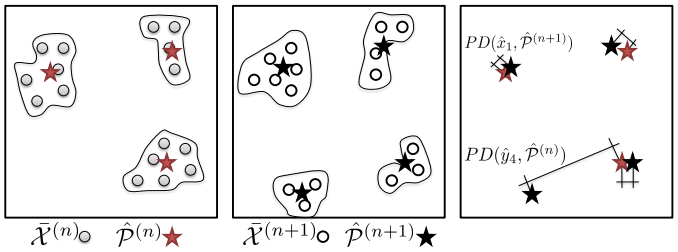


Fig. 5. Example of Clusterings, Patterns and Centroid Distance computations.

Fig. 5 depicts the *Pattern Dissimilarity* computation considering a 2-dimensional space. From left to right, DBSCAN first clusters the points (grey dots for the first snapshot, white for the second). Then, centroids emerge to form the patterns, and we compute the Centroid Distance for each centroid. Finally, the *Pattern Dissimilarity* is the sum of all Centroid Distances.

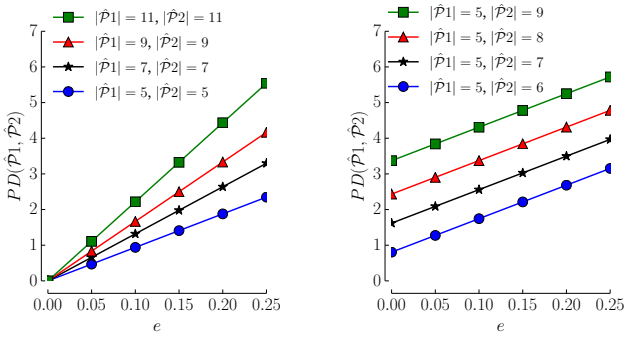
In the following, we consider two subsequent snapshots n, and $n+1$, compute the clustering $\mathcal{C}^{(n)}$ and $\mathcal{C}^{(n+1)}$, then extract the patterns $\hat{\mathcal{P}}^{(n)}$ and $\hat{\mathcal{P}}^{(n+1)}$, and finally compute their dissimilarity $PD\left(\hat{\mathcal{P}}^{(n)}, \hat{\mathcal{P}}^{(n+1)}\right)$.

As we discuss in Sec. II, to the best of our knowledge we are the first to propose an approach to quantify the similarity among different clustering results. We note that we can base the *Pattern Dissimilarity* on other similarity metrics different from the Euclidean distance, e.g., the well known Cosine Similarity. However, as we show in Sec. IV-C using the Euclidean distance lets the *Pattern Dissimilarity* to inherit linear properties, and therefore to vary proportionally with size of the changes. Observe also that the design of the *Pattern Dissimilarity* offers a nice property that is particularly desirable for troubleshooting purposes. In particular, the *Pattern Dissimilarity*, which is a simple sum of Euclidean distances, lets us immediately pinpoint the centroids responsible for changes in the pattern. As we show in Sec. VI, this aspect is crucial, as it allows us to design an automatic procedure that i) captures changes in YouTube CDN infrastructure, and ii) highlights the edge-nodes involved in these changes.

C. Observations about the Pattern Dissimilarity

We run some numerical evaluation to gauge how the *Pattern Dissimilarity* changes with respect to changes in the data. We consider two main sources of changes: i) centroids that simply move from their position, and ii) the birth of new centroid reflecting the generation of a new cluster in the data.

For the first scenario, we generate a random pattern $\hat{\mathcal{C}}1$ of $N = |\hat{\mathcal{C}}1|$ centroids. We randomly place centroids in the unitary hypercube of edge 1 in $\mathbb{R}^N$ according to a uniform distribution. Then, we generate pattern $\hat{\mathcal{C}}2$ by taking the centroids in $\hat{\mathcal{C}}1$, and repositioning them in a random sphere of radius e centered in the centroid original position. Finally, we compute $PD(\hat{\mathcal{C}}1, \hat{\mathcal{C}}2)$. We repeat the experiment for 100 times, and average the obtained values. Fig. 6(a) reports the average *Pattern Dissimilarity* for increasing values of e, and for different values of N. As expected, curves pass through the origin, and linearly grow with e. The larger is N, the higher is the average *Pattern Dissimilarity*.



(a) $|\hat{\mathcal{P}}1| = |\hat{\mathcal{P}}2|$ (b) $|\hat{\mathcal{P}}1| < |\hat{\mathcal{P}}2|$

Fig. 6. *Pattern Dissimilarity* for increasing noise e, and constant or increasing number of centroids.

For the second case, we run the same experiment while also increasing the number of centroids. Thus $|\hat{\mathcal{C}}1| < |\hat{\mathcal{C}}2|$. Fig. 6(b) shows the results. Notice the nice property of the *Pattern Dissimilarity* for which the birth of new centroids causes the *Pattern Dissimilarity* to grow by a factor that is proportional to the number of new centroids. This is due to definition in Eq.(11) in which no normalization is present. This property is important, as it lets the *Pattern Dissimilarity* nicely highlight the sudden birth (or death) of centroids.

V. Results

In this section we first assess and tune the performance of DBSCAN in order to identify edge-nodes. We next run *YouLighter* over a longitudinal dataset to show its ability to highlight sudden changes in the YouTube CDN.

A. DBSCAN performance

1) Clustering Performance Metrics: We first evaluate the impact of the parameter settings on the DBSCAN clustering results. In particular, we aim to understand how good is the matching between the clustering DBSCAN returns and the edge-nodes we observe in the measurements. To perform this analysis, we consider the snapshot X from November 4th to November 10th, 2013, in trace *ISP1-A*. We manually inspect the dataset, and, guided by the IATA codes, we assign each cache a label corresponding to the edge-node in the YouTube CDN. We manually cross-check labels by inspecting server IP addresses and subnets, RTT and TTL distributions to verify the accuracy of the labels. The result is a ground truth label, GT-label, that we assign to each cache. In total we find $|X| = 620$ caches serving more than $MinFlow = 50$ flows, and belonging to 6 edge-nodes, each identified by a different GT-label. Hence, the number of GT-labels is $N_{GT} = 6$.

We then run DBSCAN as described in Sec. IV-A, obtaining the clustering $\mathcal{C}$. Let $N_C = |\mathcal{C}|$ be the number of clusters. We next use the GT-labels to assign a label to caches by using a majority-voting scheme: For each cluster $C_j \in \mathcal{C}$, we assign all caches $x \in C_j$ the most frequent GT-label observed in C_j. Caches whose assigned label matches the GT-label are the so called True Positives (TP), whose number is N_{TP}. Conversely, caches whose assigned label is different from their GT-label are False Positives (FP), whose number is N_{FP}. $|X| = N_{TP} + N_{FP}$. We compute the set of distinct labels assigned to clusters in $\mathcal{C}$, whose number is $N_L \leq N_{GT}$. We do not assign any label to the caches which DBSCAN classifies as noise points.

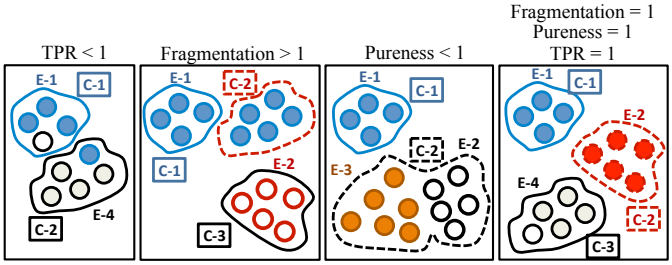


Fig. 7. Examples of patterns for which the *True Positive Rate* , the *Fragmentation Index*, and the *Pureness Index* are not equal to 1, and the optimal case in which they are all equal to 1. Color represent the GT-label.

To validate the clustering we obtain with DBSCAN, we compute the followings indices:

$$\text{TPR} = \frac{N_{TP}}{|X|}, \ \mu = \frac{N_C}{N_L}, \ \phi = \frac{N_L}{N_{GT}} \tag{12}$$

i) The *True Positive Rate* ($TPR \leq 1$) is the ratio between TP and the number of samples in the experiment. $TPR = 1$ means that all labels are identical to the GT-label. $TPR < 1$ indicates the presence of i) mislabelled caches (or FP), or ii) noise points (unlabeled points). Leftmost sub-figure in Fig. 7 reports a simple example where the clustering algorithm mislabels a cache for both the GT-labels E-1 and E-2, thus leading to $TPR < 1$. Colors represent the GT-label.
ii) The *Fragmentation Index* ($\mu \geq 1$) captures the case when more clusters share the same GT-label. When $\mu = 1$, the number of clusters is identical to the number of GT-labels and DBSCAN assigns each cluster a different GT-label. When $\mu > 1$ instead, we have more clusters which share the same GT-label, i.e., DBSCAN splits an edge-node into two or more clusters. Second sub-figure in Fig. 7 reports an example where the clustering algorithm splits edge-node E-1 in two different clusters, C-1 and C-2, thus leading to $\mu > 1$.
iii) The *Pureness Index* ($\phi \leq 1$) measures the ability to identify all edge-nodes. When $\phi = 1$, DBSCAN assigns each GT-label to at least one cluster, i.e., it correctly identifies all edge-nodes. $\phi < 1$ indicates that some edge-nodes disappear into other clusters (i.e., their GT-label is not the majority label for any cluster). Third sub-figure of Fig. 7 reports an example where the clustering algorithm groups together edge-nodes with GT-labels E-2 and E-3 in cluster C-2, thus leading to $\phi < 1$.

Rightmost sub-figure in Fig. 7 also depicts the ideal clustering result in which DBSCAN groups correctly the caches for all the edge-nodes, i.e., one cluster for each GT-label (edge-node), leading to the case in which all the clustering performance indices, TPR, μ and ϕ, are equal to 1.

Finally, we use also the number of noise points as an index of bad clustering results, i.e., the inability of DBSCAN to group caches into edge-nodes.

2) DBSCAN Performance and Parameter Sensitivity: We run experiments to evaluate the impact of DBSCAN parameters, i.e., the choice of the features, $MinPts$ and ϵ. For now, we set features as the 20th, 35th, 50th, 65th, 80th percentiles for both the RTT and TTL distributions. $MinPts$ is typically not critical since it defines the minimum number of caches in an edge-node DBSCAN needs to form a cluster. We set it to 5. Instead, we must choose ϵ carefully: If too small, a lot of fragmented clusters will emerge, or a large number of points will not be able to form dense areas, increasing the number of

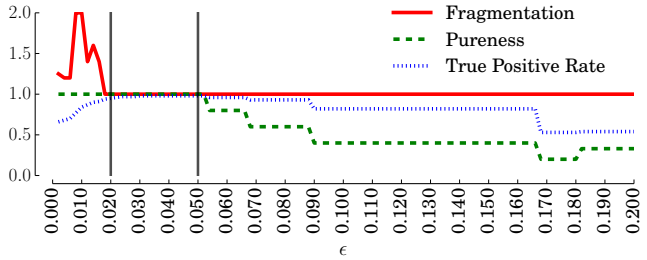


Fig. 8. DBSCAN performance versus ϵ with percentiles as feature. 1st week of November, *ISP1-A*.

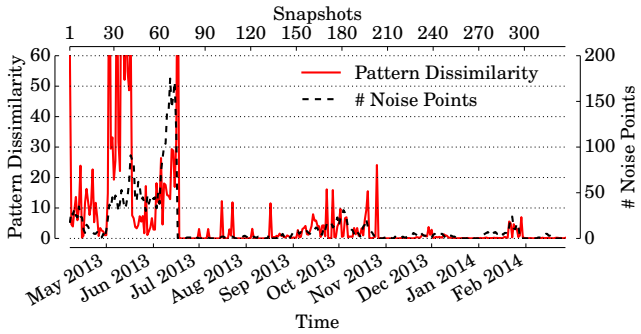


Fig. 9. *Pattern Dissimilarity* values and number of noise points for *ISP1-B*.

noise points; conversely, large values tend to create few, large clusters, that aggregates caches from different edge-nodes.

Fig. 8 reports the clustering indices when varying $\epsilon \in [0.0 : 0.2]$. As shown, we achieve the best performance with values between 0.018 and 0.052 (in between the vertical solid lines). For such values, all the three indices are equal or very close to 1. Smaller values of ϵ increase the number of noise points and artificially fragment edge-nodes into multiple clusters. TPR decreases, while μ first increases, then decreases due to caches DBSCAN labels as noise (more than 300 caches fall in the noise for $\epsilon < 0.005$). For ϵ larger than 0.052 DBSCAN merges edge-nodes into too few clusters, and both ϕ and the TPR considerably decrease. We repeat this analysis for other traces and for different snapshots. We find $\epsilon \in [0.02 : 0.045]$ to give consistent results. In the following we choose $\epsilon = 0.04$.

We also run a set of experiments to choose which features to use to capture the RTT and TTL distributions. We replace the vector of percentiles $P_m(x)$ in Eq.(3) with simple statistics, e.g., the mean and the standard deviation. The goal of this experiment is to verify whether we can replace the percentiles with some measure which does not require us to build an empirical distribution, a task which requires to collect a fairly large number of flows per cache. Unfortunately, in this case, good clustering can be obtained for a much more restricted values of ϵ, e.g., $\epsilon = 0.035$ (the plot, not reported in this paper due to space constraints, is available in [19]). By manually investigating the reasons, we observe that the mean and standard deviation varies widely among caches in the same edge-node. This variability is due to the tail of the distributions which is affected by few outliers, e.g., very large RTT samples that bias the mean and standard deviation, but have little impact on percentiles. Indeed, percentiles are very similar, except those that gauge the tail (see the 95th percentiles in Fig.4). This suggests that the choice of the percentiles to populate the vector $P_m(x)$ is more robust with respect to other simpler statistics. We run other experiments with different percentile choices that we do not report for the sake of brevity. We find no significant differences when avoiding considering percentiles in the tail. Similarly, we observe that using both RTT and TTL gives better results than considering RTT or TTL alone.

VI. YouLighter's highlighting capability

We run *YouLighter* over the four traces in Tab. I to validate its capability of highlighting changes in the YouTube CDN. The rationale is to let the ISP observe macroscopic changes that may affect a large number of users, and which may last for moderate time periods. We consider $\Delta T = 7$ days, and we start a new snapshot at midnight of every day. Snapshots form a sliding window that moves forward every day, and aggregates statistics for the past seven days. $\Delta T = 7$ days guarantees to collect large enough number of samples for the large subset of the most used caches.

Fig. 9 shows the evolution of the *Pattern Dissimilarity* (red solid curve, left y-axes) over time for *ISP1-B*. It also depicts the evolution over time of the number of caches that remain in the noise after clustering (black dashed curve, right y-axes). The plot refers to *ISP1-B*. X-axes reports daily snapshots, starting from April 1st, 2013. We compute *Pattern Dissimilarity* on *ISP1-A* and *ISP1-C* too. Results, omitted here due to space constraints, are available in our technical report [19].

As shown, the *Pattern Dissimilarity* is very good at highlighting events. Indeed, according to Sec. IV-C, a $PD > 10$ suggests that the clustering at time (n) is very different to the one at time $(n+1)$. Thanks to the data aggregation we obtain with the clustering, we can easily analyze the highlighted events, and quickly identify the edge-nodes involved in the changes. We investigate these events, and verify that they all correspond to sudden changes in the edge-nodes used by YouTube in serving ISP customers. In the following, we illustrate the most relevant ones, i.e., those with a $PD > 50$.

A. Large event, involving all ISP customers

We first investigate an event *YouLighter* highlights in different datasets. It starts on May 2nd (snapshot 27), May 7th (snapshot 32) for *ISP1-B* and *ISP1-A*, respectively. *Pattern Dissimilarity* peaks above 60. Starting from then, both PD and the number of noise points are very large. This indicates an unstable behavior, with many caches that DBSCAN cannot successfully group together, and the clustering pattern that keeps changing day by day, for more than 40 days.

To give the intuition of what happened, Fig. 10 shows the per-cache percentiles of the RTT that we measure in *ISP1-A* before, during, and after the anomalous event. First, we notice that most of the edge-nodes suddenly change: E-1, E-4, E-5 and E-6 actually "disappear" from the clustering pattern, and during the event, many previously unseen caches in edge-node E-2 start serving lots of customers (observe the center plot). Second, and more surprisingly, the path properties to these new caches is by far different from paths to other caches in E-2: the RTT percentiles are much larger (95ms versus 15ms for the 50th percentile) and much more variable. Despite these caches share the same IATA code (E-2), the path to reach them is different from the path of other caches in E-2, with the former possibly being severely congested. Some of these caches form new clusters, but most of them become part of the noise: Indeed, their features do not correspond to the

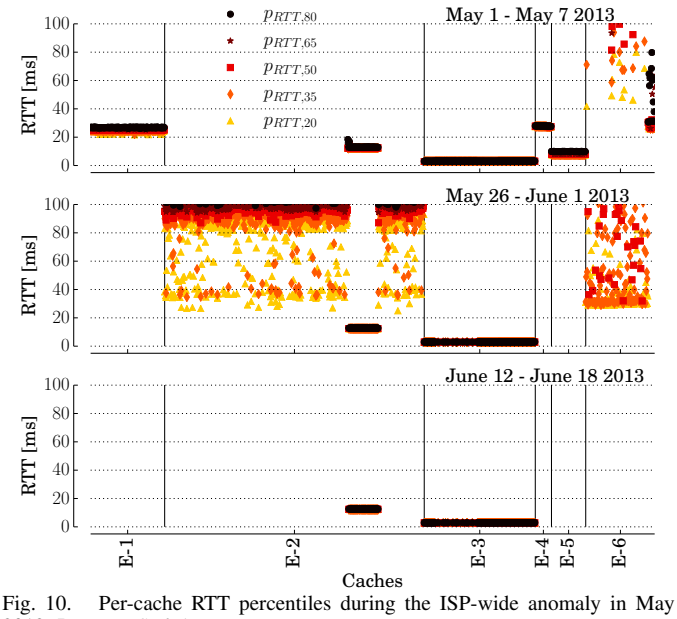


Fig. 10. Per-cache RTT percentiles during the ISP-wide anomaly in May 2013. Dataset *ISP1-A*.

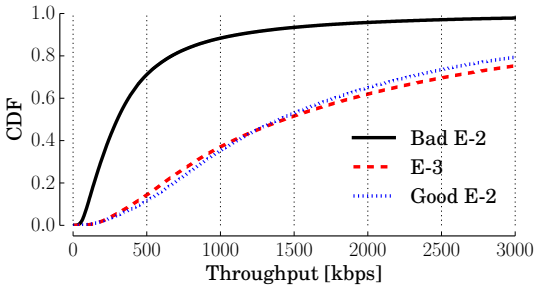


Fig. 11. Throughput distribution for flows served by E-3, Good-E-2 and Bad-E-2 during the large anomaly we observe in May 2013. Dataset *ISP1-B*.

Format	Good-E2	Bad-E2
144p	17.4%	31.7%
240p	18.3%	26.1%
360p	45.4%	35.7%
480p	14.5%	5.3%
720p	3.8%	1.0%
1080p	0.6%	0.2%
AAC128	80.3%	92.0%
AAC256	19.7%	8.0%

TABLE II. FRACTIONS OF VIDEO AND AUDIO DASH FORMATS SERVED BY GOOD-E-2 AND BAD-E-2. DATASET *ISP1-B*.

ones DBSCAN's tuning is expecting, i.e., the distance between points is higher than $\epsilon = 0.04$. We call these caches Bad-E-2, in opposition to the small share of caches still belonging to E-2, but showing small RTT, i.e., Good-E-2.

We now analyze the impact of such change on the Quality of Experience the ISP customers perceive. We report in Fig. 11 the distributions of the download throughput obtained by video retrieved by caches in E-3, the best edge-node to ISP customers, Good-E-2 and Bad-E-2. The difference is striking: while videos served by E-3 and Good-E-2 have throughput that allows to enjoy YouTube with no major impact on the QoE (>1,000 kb/s in 63% of the cases), the throughput for Bad-E-2 caches is below 500 kb/s (250 kb/s) in 75% (40%) of the cases, clearly not enough to enjoy a video with a satisfiable QoE. Tab. II corroborates above observation reporting the fractions of video (and audio) formats seen in flows handled by both Good-E-2 and Bad-E-2.[7] For this analysis we consider only DASH formats, as for these formats the cache delivering the video automatically adapts the quality of the video stream depending on the congestion it measures on the path to the client. As shown, Good-E-2 serves larger fractions of high-definition videos. Conversely, the share of videos encoded with low-definition (144p and 240p) increases for Bad-E-2. This confirms that Bad-E-2 experienced possible congestion during the monitored period, severely impairing the QoE of the users.

By double checking this event with the ISP network support team, we confirm the incident involved most of their customers, increasing dramatically the complaining at their customer support. This confirms the pervasiveness of this event upon ISP customers.

B. Other events for ISP1

We manually cross check other events, and find that some of those affected only part of the ISP customers. This shows that YouTube CDN allocates customers to edge-nodes using a fine grained granularity, i.e., the load-balancing allows to identify small groups of clients by using the client IP address (or network). For instance, on October 2nd (snapshot 180) and October 9th (snapshot 187) *YouLighter* highlights two sudden changes in the *ISP1-A*, as the *Pattern Dissimilarity* peaks over 60. Inspecting the point dissimilarity one by one, we observe that the changes are due to 3 edge-nodes (E-4, E-5 and E-6) out of 7 that suddenly "appear" in snapshot 180 and "disappear" in snapshot 187. The remaining four edge-nodes then serve the videos for customers in *ISP1-A*. We analyze the impact of the presence of such caches on the QoE by measuring the aggregate download throughput before, during and after their permanence, but we do not appreciate any significant change. Also in this case we double check the event with the ISP support team and we confirm that the change had no influence on the QoE as the customer support did not receive any meaningful complaining in the considered period.

Finally, for *ISP1-B*, we do not detect any change ($PD = 0.12$) in the same period, as YouTube's CDN keeps serving customers with the same group of edge-nodes.

C. Events in ISP2

As a last set of experiments, we run *YouLighter* on the *ISP2* dataset, which we collect in a different ISP in a different country. We run *YouLighter* with the same parameters we tune for ISP1, i.e., without going through ϵ optimization. Indeed we aim to check whether if the edge-node model that DBSCAN creates is general and robust enough to work in a completely different scenario.

We repeat the experiment of Fig. 9 for *ISP2* dataset, and we analyze the evolution of the *Pattern Dissimilarity* and number of noise points. We report the results in Fig. 12. To check if the clustering correctly identifies the edge-nodes, we select five different snapshots at random among the ones where *YouLighter* highlights no events. Again, we use the IATA codes as ground truth, and we manually check IP address subnets, RTTs and TTLs to see if some suspicious cache is present

[7]Observe that in our dataset only a tiny portion (∼1%) of requests are HTTPS, and, thus, encrypted. For the wide majority of the cases,the information about video and audio formats are exposed in plain text in HTTP requests.

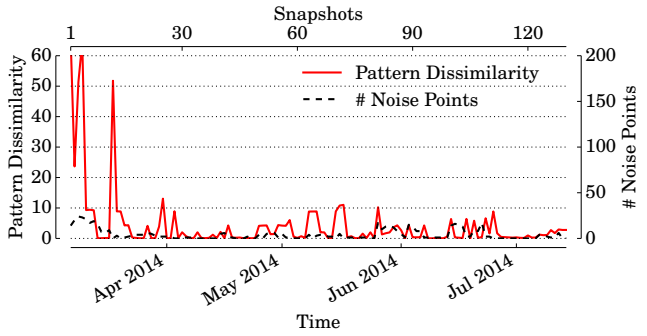


Fig. 12. *Pattern Dissimilarity* values and number of noise points for dataset *ISP2*.

in a cluster. The clustering results in perfect match with the (possible) edge-nodes in the ground truth. This despite edge-nodes, path, and ISP in this dataset are completely different.

We then check two suspicious events. The first one occurs from March 7th to March 10th, 2014 (snapshots 1-4, $PD > 60$), and the second one happens on March 18th, 2014 (snapshot 12, $PD > 51$). We observe that the first anomaly is due to a change in the network path to reach a small group of caches in E-2. We observe that this deviation does not influence the QoE perceived by the users. For the second event, by comparing the clustering at snapshot 12 with the following one (March 19th), we observe a notable change in the infrastructure of the YouTube CDN: all caches belonging to a specific edge-node in E-7 disappear during this period (more details available in our technical report [19]). Also in this case, the change has no evident impact on users' QoE, as the average download throughput does not vary. However, we notice that the edge-node E-7 represents a much more expensive route for the ISP2, since it is located in an remote ISP for which no peering agreements are in place.

VII. Conclusions

In this paper we proposed a novel system, named *YouLighter*, that leverages passive observation of network traffic and unsupervised machine learning techniques to automatically monitor and identify changes in the YouTube CDN. Based on the well known DBSCAN clustering algorithm, *YouLighter* can automatically group thousands of caches into few edge-nodes. To then compare the results of clustering obtained considering the snapshots collected in consecutive time intervals, we propose the *Pattern Dissimilarity*, a novel framework that, for the first time to the best of our knowledge, allows to easily pinpoint changes in clusters.

We validated *YouLighter* using a large set of traces reporting the activity of users accessing YouTube. Our results are excellent: after a short and simple tuning procedure to find the best setup for DBSCAN, *YouLighter* could detect anomalous events that happened in YouTube CDN. For instance, we noticed a large transformation in a crucial edge-node of YouTube CDN which notably impaired the QoE perceived by the ISP customers for more than 40 days.

We believe that *YouLighter* may represent a promising opportunity for ISPs, network administrators, developers and researchers to monitor the traffic generated by YouTube CDN. ISPs, for instance, may employ *YouLighter* to design automatic traffic engineering policies or to promptly react when changes in YouTube CDN impair the QoE of their customers.

Our ongoing efforts are focused on three directions: First, we are working to automate the tuning of *YouLighter*'s parameters, and, thus, its whole operation process. Second, we are developing an online deployment of *YouLighter*, capable of detecting changes in YouTube CDN in real time. Third, we are adapting it to consider other use cases.

References

[1] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan, "Mapping the expansion of google's serving infrastructure," in *ACM IMC*, 2013.

[2] V. Adhikari, S. Jain, Y. Chen, and Z.-L. Zhang, "Vivisecting youtube: An active measurement study," in *IEEE INFOCOM*, 2012.

[3] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. Munafo, and S. Rao, "Dissecting video server selection strategies in the youtube cdn," in *IEEE ICDCS*, 2011.

[4] P. Casas, P. Fiadino, and A. Bär, "Understanding http traffic and cdn behavior from the eyes of a mobile isp," in *PAM*, 2014.

[5] P. Casas, A. D'Alconzo, P. Fiadino, A. Bar, A. Finamore, and T. Zseby, "When youtube does not work: Analysis of qoe-relevant degradation in google cdn traffic," *Network and Service Management, IEEE Transactions on*, vol. 11, no. 4, pp. 441–457, Dec 2014.

[6] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *ACM KDD*, 1996.

[7] T. Hossfeld, R. Schatz, E. Biersack, and L. Plissonneau, "Internet video delivery in youtube: From traffic measurements to quality of experience," in *Data Traffic Monitoring and Analysis*. Springer, 2013, vol. 7754, pp. 264–301.

[8] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," in *Computer Networks*, 2007, vol. 51, pp. 3448 – 3470.

[9] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," in *Computing Surveys*. ACM, 2009, vol. 41, pp. 1 – 58.

[10] H. Yan, A. Flavel, Z. Ge, A. Gerber, D. Massey, C. Papadopoulos, H. Shah, and J. Yates, "Argus: End-to-end service anomaly detection and localization from an isp's point of view," in *IEEE INFOCOM*, 2012.

[11] G. Münz, S. Li, and G. Carle, "Traffic anomaly detection using k-means clustering," in *GI/ITG Workshop MMBnet*, 2007.

[12] R. D. Torres, M. Y. Hajjat, S. G. Rao, M. Mellia, and M. M. Munafo, "Inferring undesirable behavior from p2p traffic analysis," in *SIGMETRICS Performance Evaluation Review*. ACM, 2009, vol. 37, pp. 25 – 36.

[13] M. K. Goldberg, M. Hayvanovych, and M. Magdon-Ismail, "Measuring similarity between sets of overlapping clusters," in *IEEE SocialCom*, 2010.

[14] P. Kalnis, N. Mamoulis, and S. Bakiras, "On discovering moving clusters in spatio-temporal data," in *Advances in spatial and temporal databases*. Springer, 2005, vol. 37, pp. 364 – 381.

[15] Z. Li, B. Ding, J. Han, and R. Kays, "Swarm: Mining relaxed temporal moving object clusters," in *Proceedings of the VLDB Endowment*. VLDB Endowment, 2010, vol. 3, pp. 723–734.

[16] D. Kifer, S. Ben-David, and J. Gehrke, "Detecting change in data streams," in *ACM VLDB*, 2004.

[17] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the s in https," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 2014, pp. 133–140.

[18] I. N. Bermudez, M. Mellia, M. M. Munafo, R. Keralapura, and A. Nucci, "Dns to the rescue: Discerning content and services in a tangled web," in *IMC*, ACM, Ed., 2012.

[19] D. Giordano, S. Traverso, L. Grimaudo, M. Mellia, E. Baralis, A. Tongaonkar, and S. Saha, "Youlighter: An unsupervised methodology to unveil youtube cdn changes," *ArXiv e-prints*, Mar. 2015.