

YouLighter: A Cognitive Approach to Unveil YouTube CDN and Changes

Danilo Giordano, Stefano Traverso, *Member, IEEE*, Luigi Grimaudo, Marco Mellia, *Senior Member, IEEE*, Elena Baralis, *Member, IEEE*, Alok Tongaonkar, *Senior Member, IEEE*, and Sabyasachi Saha

(Invited Paper)

Abstract—YouTube relies on a massively distributed content delivery network (CDN) to stream the billions of videos in its catalog. 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 almost no visibility and understanding of CDN decisions. This paper presents *YouLighter*, an unsupervised technique that builds upon cognitive methodologies to identify changes in how the YouTube CDN serves traffic. *YouLighter* leverages only passive measurements and clustering algorithms to group caches that appear colocated and identical into edge-nodes. This automatically unveils the YouTube edge-nodes used by the ISP customers. Next, we leverage a new metric, called *Pattern Dissimilarity*, that compares the clustering results obtained from two different time snapshots to pinpoint sudden changes. By running *YouLighter* over 10-month long traces obtained from two ISPs in different countries, we pinpoint both sudden changes in edge-node allocation, and small alterations to the cache allocation policies, which actually impair the QoE that the end-users perceive.

Index Terms—Network Monitoring, Clustering, YouTube.

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]. It consists of hundreds of edge-nodes scattered in the Internet. Each edge-node hosts hundreds

Manuscript received June 18, 2015; revised November 18, 2015; accepted December 21, 2015. This work was supported by the European Union under the FP7 under Grant 318627 (Integrated Project “mPlane”) and in part by Narus Inc., Sunnyvale, CA. A preliminary version of this paper has been presented at the 27th International Teletraffic Congress (ITC27). The associate editor coordinating the review of this paper and approving it for publication was M. Zorzi.

D. Giordano, S. Traverso, L. Grimaudo, M. Mellia, and E. Baralis are with the DET Dipartimento di Elettronica e Telecomunicazioni, Politecnico di Torino, Turin 10129, Italy (e-mail: first.last@polito.it).

A. Tongaonkar is with the Symantec Corporation, Mountain View, CA 94043 USA (e-mail: alok_tongaonkar@symantec.com).

S. Saha is with the Cyphort, Santa Clara, CA 95054 USA (e-mail: sabyasachi.saha@gmail.com).

Digital Object Identifier 10.1109/TCCN.2016.2517004

¹<https://www.youtube.com/yt/press/statistics.html>

of video servers, or *caches*, which can each potentially serve any video any 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 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 the customers 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, the YouTube CDN allocation policy frequently changes caches being used to serve videos, and changes 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, or an eventual change on the path to those caches, e.g., due to congestion, or route change. 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.

A. Our Contribution

In this paper, we present *YouLighter*, a novel methodology to automatically monitor and pinpoint changes in how the YouTube CDN serves traffic. *YouLighter* relies on an unsupervised learning approach that, as such, does not require any knowledge of the YouTube infrastructure. It builds upon a cognitive approach, where automatic and unsupervised algorithms are used to extract a model of the system status. *YouLighter* only assumes that the ISP has deployed passive traffic 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

²<https://peering.google.com/about/index.html>

79 *snapshot* of the traffic exchanged with YouTube caches. Based
80 on DBSCAN [6], a well-established unsupervised machine
81 learning algorithm, *YouLighter* is able to automatically group
82 thousands of caches into a bunch of edge-nodes using simple
83 features that characterize the network distance of caches from
84 the vantage point.

85 *YouLighter* periodically runs DBSCAN on consecutive snap-
86 shots, extracting for each of them a model of the status of the
87 CDN. The problem becomes then how to compare the two mod-
88 els to highlight eventual changes. We solve it with some inge-
89 nuity, we summarize the corresponding models into *patterns*,
90 and compare them using the notion of *Pattern Dissimilarity*, a
91 metric similar to others presented in the literature, but which
92 satisfies our specific requirements (see Sec. II for a detailed
93 discussion). The bigger the distance between two snapshots
94 is, the more different the sets of YouTube caches to serve ISP
95 customers during the two periods of time are.

96 *YouLighter* highlights several kinds of changes, including
97 deviations from the typical behavior of edge-nodes possibly
98 induced by congestion arising in the network. In general,
99 *YouLighter* triggers alarms corresponding to sudden changes
100 happening in the YouTube CDN infrastructure which may be
101 responsible of QoE issues for ISP customers. Resulting alarms
102 are then offered to the ISP network administrator who can take
103 countermeasures to mitigate the problem.

104 We validate our methodology over traces we collect from
105 four different vantage points that we have deployed in two
106 ISPs in two different countries. First, we demonstrate that the
107 cognitive algorithms *YouLighter* adopts are effective at iden-
108 tifying and grouping YouTube caches belonging to different
109 edge-nodes. Second, we run *YouLighter* over different collected
110 snapshots considering a longitudinal dataset, which, overall,
111 accounts for more than 33 months of traffic. We pinpoint sev-
112 eral examples of sudden and previously undiscovered changes
113 in the YouTube CDN. For some of them, we investigate the
114 impact on the QoE of ISP customers, revealing the sudden drop
115 of average video download throughput to less than 250 kb/s,
116 which hampers even the possibility of watching a video.

117 We believe that *YouLighter* is a promising tool for ISPs, net-
118 work administrators and researchers to monitor the YouTube
119 CDN and the traffic it generates. Importantly, thanks to its
120 design, *YouLighter* offers the capability of automating and
121 accelerating the troubleshooting procedures. In fact, ISPs may
122 use *YouLighter* to quickly react to changes possibly harming
123 customer's QoE. For instance, ISPs may adopt traffic engi-
124 neering algorithms to optimize routing to under-performing
125 edge-nodes, e.g., by means of BGP policies, or to implement
126 DNS policies overruling YouTube choices and re-directing traf-
127 fic from caches with bad QoE to changes with a good QoE.
128 However, *YouLighter*'s task is limited to notifying the occur-
129 rence of change events in the YouTube CDN. The investigation
130 and troubleshooting of issues notified by *YouLighter* are out of
131 the scope of this paper.

132 The remainder of this paper is structured as follows: Sec. II
133 discusses the related work. Sec. III describes the details of our
134 datasets, and shows the dynamicity of YouTube cache selec-
135 tion policies. Sec. IV presents our methodology, introduces
136 the notion of *Pattern Dissimilarity*, and discusses *YouLighter*'s
137 complexity. Sec. V presents our results: First, we evaluate

the sensitivity of *YouLighter*'s parameters, and, second, we 138
show how effective *YouLighter* is at pinpointing changes in 139
YouTube CDN employing our traces. Sec. VII suggests some 140
countermeasures an ISP can use to improve users' QoE in case 141
of changes. Finally, Sec. VIII concludes the paper. 142

II. RELATED WORK 143

A large body of work has analyzed the YouTube delivery 144
infrastructure and its evolution over time [1]–[3], [5], [7]. They 145
show YouTube is a highly dynamic system which keeps chang- 146
ing over time due to continuous upgrades in the infrastructure 147
[1], [2] or due to the dynamicity of the cache selection policies 148
[3], [7]. Some of the findings are already outdated. For instance, 149
the load-balancing policy based on HTTP redirections which is 150
described in [3], [7] is no longer in place, and YouTube dis- 151
missed the naming scheme described in [2] at the end of 2011. 152
In this work, we do not aim to offer an updated view or charac- 153
terization of YouTube. Instead, we present a methodology that 154
allows to automatically identify changes in both the infrastruc- 155
ture, e.g., the appearance of new edge-nodes, and in the day 156
to day management of the infrastructure, e.g., a change in the 157
load-balancing algorithm that may affect millions of customers. 158

Our contribution is in line with the body of works focusing 159
on anomaly detection, for which [8], [9] offer good surveys. 160
In particular, our work belongs to the family of studies which 161
addresses the problem of performing anomaly detection in large 162
scale operational networks. [10], [11] are notable examples 163
of supervised methodologies which leverage data from pas- 164
sive probes, topology information, routing tables and Simple 165
Network Management Protocol (SNMP) logs to match predic- 166
tions to actual measurements to pinpoint deviations. 167

Other works propose methodologies to perform anomaly 168
detection on CDN infrastructures for video delivery specifi- 169
cally [5], [12], [13]. Authors of [12] consider a collection of 170
video download sessions, out of which they extract measure- 171
ments for specific features, and manually set thresholds to label 172
degradation-affected sessions. Then, by applying graph tech- 173
niques, they identify clusters and outliers possibly associated 174
to performance issues. [13] analyzes different CDN providers, 175
among which YouTube. The paper presents a characterization 176
of the YouTube cache selection policy and apply an anomaly 177
detection system based on subnet usage to detect unexpected 178
cache selection in a time window of minutes. Finally, [5] 179
focuses on the YouTube case too and proposes a methodol- 180
ogy for anomaly detection which requires a significant manual 181
effort, and the paper mostly presents results about the charac- 182
terization of the YouTube service in terms of traffic characteristics 183
and QoE perceived by the users. *YouLighter* differs from the 184
supervised methodologies described in above studies. In fact, 185
YouLighter does not assume any knowledge of a baseline, 186
and leverages unsupervised algorithms to automatically unveil 187
changes on the YouTube infrastructure. We design it with this 188
specific requirement in mind, as it has to target the YouTube 189
CDN, for which the ground truth is a moving target that is very 190
difficult to know. 191

The application of cognitive and unsupervised machine 192
learning techniques – in particular clustering techniques – to 193
perform anomaly detection based on network traffic is not new. 194

195 For instance, [14] proposes a flow-based anomaly detection
 196 algorithm based on K-Means, while [15] uses DBSCAN to
 197 group P2P sessions and identify anomalous clusters. However,
 198 in all the cases, clustering is used to study the same given
 199 dataset. Our goal is anomaly detection algorithm builds on the
 200 comparison between clustering patterns obtained at different
 201 times. The task of identifying anomalies by comparing cluster-
 202 ing results attained from different datasets (e.g., different
 203 time snapshots, etc.) translates in the problem of measuring
 204 the dissimilarity of two distinct patterns. The comparison
 205 and quantification of the similarity obtained from evolving-
 206 in-time clustering is addressed within a sub-domain referred
 207 to as *online clustering* or *streaming clustering*. See [16], [17]
 208 for notable examples. In particular, [16] defines a metric to
 209 quantify the dissimilarity between consecutive clustering pat-
 210 terns, namely the History Cost, which is similar to the Pattern
 211 Dissimilarity presented in this work. However, the History Cost
 212 is designed to compute the distance between clustering patterns
 213 built by K-Means, for which the number of clusters is con-
 214 stant across different patterns. Instead, the Pattern Dissimilarity
 215 takes into account the case in which the number of clusters
 216 across different patterns is different. Another work which goes
 217 in a similar direction is [18] whose authors propose to measure
 218 similarity between sets of overlapping clusters from complex
 219 networks, in which groups of nodes form tightly connected
 220 units linked to each other. Since points are not embedded in
 221 a metric space, authors of [18] define ad-hoc distances.

222 *YouLighter* differs also from techniques for the tracking of
 223 moving clusters and objects as in [19], [20]. Indeed, their goal is
 224 to track the movements of the same clustered objects over time,
 225 e.g., a group of migrating animals. On the contrary, *YouLighter*
 226 has no insights about the CDN infrastructure and it cannot track
 227 single objects, which may disappear and reappear freely.

228 Finally, other approaches as [21] measure the similarity
 229 among sample distributions obtained at different time inter-
 230 vals. However, directly relying on distributions to perform
 231 the comparison considerably complicates the detection of the
 232 edge-nodes behind the changes. And from the datamining com-
 233 munity evolutionary clustering techniques such as [16], [17]
 234 have been used to study how clustering results evolve over
 235 time. Instead, *YouLighter* extracts and compares clustering pat-
 236 terns, which are simpler to process in an automatic manner, and
 237 allow to immediately pinpoint the edge-nodes (i.e., the clusters)
 238 responsible for possible deviations.

239 A preliminary version of this work has been presented [22].
 240 In this extended version we present more thorough performance
 241 analysis and sensitivity study of *YouLighter* algorithms, present
 242 a complexity evaluation and discuss possible countermeasures
 243 ISP can take to mitigate eventual problems.

244 III. DATASETS

245 We assume the ISP has instrumented the network with pas-
 246 sive probes, which collect statistics from traffic flows carrying
 247 YouTube videos. In this work, we rely on passive probes run-
 248 ning Tstat³ that we install in Points-of-Presences (PoPs) of

³<http://www.tstat.polito.it>

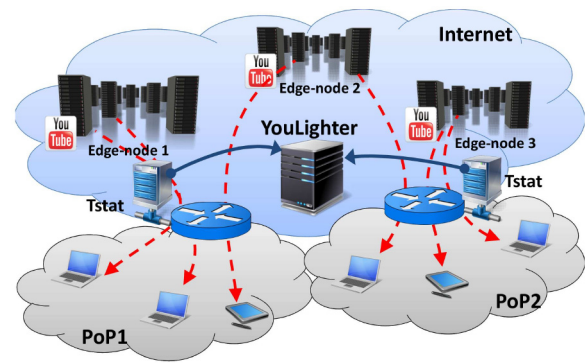


Fig. 1. The traffic monitoring setup we employ for this paper.

TABLE I
TRACES CONSIDERED IN THIS STUDY

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

operational networks, as depicted in Fig. 1. Clients are located 249
 in one PoP, and connect to the backbone via a router, where 250
 Tstat monitors the traffic. Tstat observes packets, rebuilds each 251
 TCP flow, tracks it, and at the end of the flow, logs detailed 252
 statistics. Tstat can classify TCP flows that carry YouTube 253
 videos. For each request, it logs i) the anonymized client IP 254
 address, ii) the server IP address, iii) the hostname of the server, 255
 iv) the TCP minimum Round Trip Time (RTT), v) the IP Time- 256
 To-Live (TTL) of packets received by the client in the PoP, 257
 vi) the amount of bytes the clients send and receive, vii) the 258
 average download throughput, and viii) the time at which the 259
 TCP connection starts.⁴ For more information we address the 260
 reader to [23]. Note that Tstat can compute all these metrics 261
 considering only TCP segments, and do not require access to 262
 application payload. On the one hand, this avoids any privacy 263
 issues. And other hand, it allows us to collect all needed statis- 264
 tics even in presence of encryption, e.g. when HTTPS is used, 265
 which nowadays represents more than 50% of overall YouTube 266
 traffic [24]. 267

268 We have been collecting traffic logs since April 2013 by mon- 269
 itoring the traffic users generate when accessing the Internet. 270
 We instrument four different PoPs. Three of them are located 271
 in networks of the same ISP, and in two different cities of the 272
 same country. We install the fourth one in a PoP of a differ- 273
 ent ISP in a second country. Tab. I describes, for each trace 274
 (or dataset), the time period, the total downloaded volume, the 275
 number of unique videos and the number of YouTube servers 276
 we observe. Notice that in total we monitor the activity of more 277
 than 32,000 customers, and the maximum number of YouTube 278
 caches that ISP1 customers used at least once is ~9, 000. The 279
 dataset overall covers more than 500 TB of equivalent data.

⁴The RTT is measured as the time difference between the server acknowl-
 edgement segment and the corresponding client data segment. Give a TCP
 connection, the minimum RTT is computed among all valid samples. The TTL
 is directly extracted from the IP header. Among all packets in a flow, we take
 the minimum value seen.

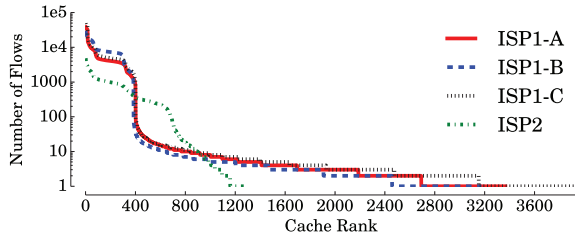


Fig. 2. Rank of YouTube caches based on the number of flows. February 2014, *ISP1*.

280 A. YouTube Cache Naming Structure

281 We find that the YouTube infrastructure described in [2]
 282 is no longer in use. During 2012, YouTube server hostnames
 283 were in the form $rx\text{---}ABCxxtxx.c.youtube.com$, where
 284 x are numbers, while ABC is a three-letter code reporting
 285 the International Air Transport Association (IATA) code of the
 286 closest airport. For instance $r7\text{---}fra07t16.c.youtube.com$
 287 identified a single cache, in Frankfurt. The hostname still
 288 resolves to a single IP address, 74.125.218.182 in the exam-
 289 ple. Thus, we can uniquely identify a cache by its hostname.⁵
 290 All caches co-located in the same edge-node share the same
 291 (obfuscated) IATA code. This allows us to get coarse ground
 292 truth about the location of servers.

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

298 B. Characterization of the Load Balancing Policies

299 Every time a user starts a video playback, the player starts
 300 a progressive download of the video content from the specific
 301 cache the system provides in the HTML page.⁶ We are inter-
 302 ested in seeing which are the policies governing the server
 303 allocation, such as (i) is there any “preferred” group of caches?
 304 or (ii) are those stable over time?

305 Fig. 2 reports the rank of YouTube caches sorted by the
 306 number of flows they serve. We consider February 2014 from
 307 the *ISP1* datasets. First, notice that we observe up to 3,800 in
 308 *ISP1-C*, with *ISP1-A* reaching more than 2,500. Second, the
 309 load each cache handles is very heterogeneous in all datasets;
 310 few servers handle lots of requests, but there is a not negligible
 311 number of caches that serves a significant portion of flows. In
 312 all three datasets, top 400 caches serve more than 1,000 videos,
 313 and, to observe 95% of requests, one must consider about 313,
 314 330, 342 caches in *ISP1-B*, *ISP1-A*, *ISP1-C* respectively.

315 We also notice that the rank is extremely dynamic over time.
 316 For instance, we pick randomly four caches among the most

⁵Starting from January 2013, YouTube obfuscates the IATA code using a simple substitution cipher. The script to de-obfuscate YouTube encrypted cache names is available at http://tstat.polito.it/svn/software/tstat/trunk/scripts/decode_yt_sitename.pl. 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 HTTPS [25].

⁶Load 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.

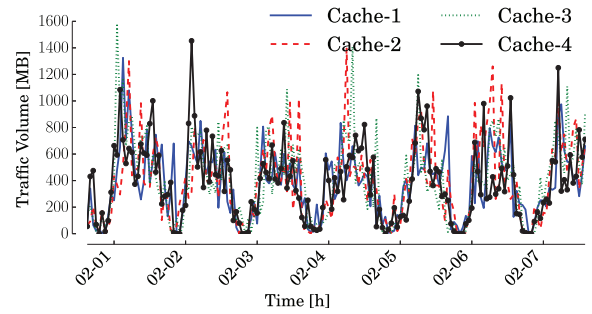


Fig. 3. Evolution of the volume of traffic for the four most active caches we observe on February 1st 2014. First week of February 2014, dataset *ISP1-A*.

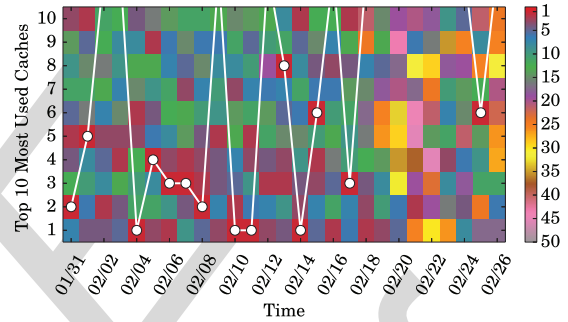


Fig. 4. Evolution over time of the rank of the top 10 mostly used caches during February 2014, *ISP1-A*. The white dot corresponds to the top cache of each day.

active caches in *ISP1-A* during the 1st week of February 2014. 317
 We report in Fig. 3 the amount of traffic they generate over time 318
 for the following seven days. As shown, the amount of traffic a 319
 single cache handles changes widely over time, and none of the 320
 monitored caches keeps a constant leading position for a long 321
 period of time. 322

As one may expect this dynamicity to disappear when reduc- 323
 ing the focus, we monitor a larger pool of caches as those in 324
 the rank in Fig. 2, and we recompute the same rank on a daily 325
 basis. Then, we represent it using different colors in Fig. 4. 326
 Each row represents the rank of the same cache for different 327
 days in February. In case the rank is stable, one would expect a 328
 row (a cache) to always assume the same color (rank). Fig. 4 329
 shows exactly the opposite. Indeed the top daily cache (red 330
 square, highlighted by the white dot) randomly changes every 331
 day (white line). Sometimes, the most used cache in a day is not 332
 among the top-10 cache of the month (line jumps outside). The 333
 top-10 caches in the monthly rank drops below the 50th place 334
 during some days (gray color). Similarly, in the first 19 days 335
 of February, the top-10 caches are concentrated in the first 20 336
 rankings; However, starting from February 20th they fall around 337
 the 30th position (notice the concentration of yellow and orange 338
 boxes). 339

This shows that the server allocation policies adopted by 340
 YouTube spread the load over several hundreds of caches, and 341
 the choices are extremely dynamic over time if we observe with 342
 the fine grained granularity of a single cache. 343

Since caches inside the same edge-node are all equivalent, 344
 the intuition is to observe the system using the coarse granular- 345
 ity offered by edge-nodes. However, edge-nodes are unknown, 346

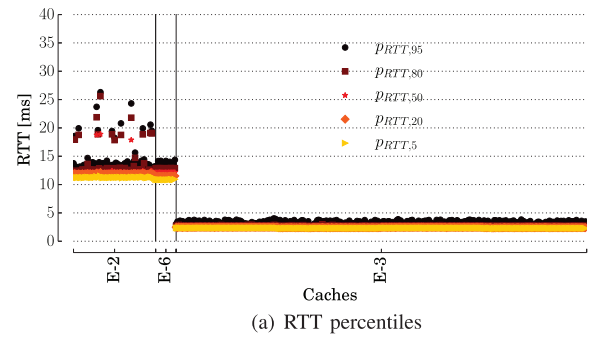
347 they can change over time due to system upgrade, maintenance
 348 operations, or redesign. Information that could be available
 349 (e.g., the IATA code) may be not reliable, or may be outdated
 350 by YouTube. This calls for cognitive systems that can automat-
 351 ically identify presence of edge-nodes and to build a model
 352 of the current status of the CDN. We thus design an unsu-
 353 pervised machine learning algorithm to automatically identify
 354 edge-nodes from just the observation of traffic flows.

355

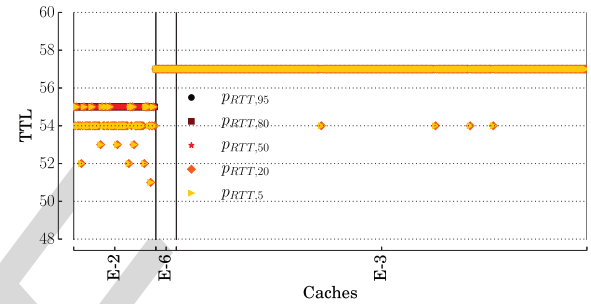
IV. METHODOLOGY

356 Among cognitive approaches that try to extract models out of
 357 data, cluster analysis (or clustering) represents a well known set
 358 of unsupervised algorithms that have been successfully used in
 359 the literature. Clustering algorithms group objects with similar
 360 characteristics [26]. Objects are described by means of features
 361 which map each object to a specific position in a hyperspace.
 362 The similarity between two objects is based on their *distance*.
 363 The closer the two objects are, the more likely they are similar
 364 and thus should to be grouped in the same cluster. Typically, the
 365 Euclidean distance is used.

366 The selection of the features plays a key role in the design of
 367 clustering algorithms. In our scenario we would like to group
 368 together caches based on the network position as seen from
 369 a vantage point. Intuitively, the path between two caches in
 370 the same edge-node and clients in the same PoP exhibits the
 371 same properties, that from a network perspective translates in
 372 same RTT and TTL. Conversely, the paths between two caches
 373 in two different edge-nodes should present different RTT or
 374 TTL. To confirm this intuition we perform the following experi-
 375 ment. We consider three edge-nodes observed in *ISPI-A* during
 376 December 12-18 2013. For each cache, we consider all the
 377 flow directed to it and we compute the Cumulative Distribution
 378 Function (CDF) of the RTT and the TTL extracted from the
 379 flows. Then we summarize each CDF with the 5th, 20th, 50th,
 380 80th, and 95th percentiles, and report the result in Fig. 5. The
 381 x axis reports caches grouped together by (anonymized) IATA
 382 code. On the y axis, for each cache, we report the percentile
 383 values, shaded with different colors, obtained from the per-
 384 cache CDFs. Fig. 5(a) and Fig. 5(b) reports the results for RTT
 385 and TTL, respectively. Since we sort caches based on their IP
 386 address and IATA code, caches belonging to the same edge-
 387 node appear one close the other. Three edge-nodes are present,
 388 E-2, E-6, and E-3. Each hosts a variable number of caches,
 389 with E-3 being the largest. As shown in Fig. 5(a), the caches
 390 in the same edge-node exhibits very similar RTT percentiles,
 391 suggesting that we can identify clusters of caches by consider-
 392 ing the RTT as a feature. However, E-2 and E-6 exhibit very
 393 similar RTT percentile values. This fact clearly complicates the
 394 capability of a clustering algorithm to label them as different
 395 edge-nodes. Similarly, when focusing on the TTL (Fig. 5(b))
 396 we observe that E-6 and E-3 show practically identical values.
 397 However, if we put Fig. 5(a) on top of Fig. 5(b), we can
 398 easily distinguish the three edge-nodes as they all exhibit differ-
 399 ent RTT and TTL combinations, and E-2, E-3 and E-6 emerge
 400 clearly as different edge-nodes. This simple example justifies
 401 our choice to consider the multidimensional space obtained by
 402 combining percentiles for both RTT and TTL features.



(a) RTT percentiles



(b) TTL percentiles

Fig. 5. Example of per-cache RTT percentiles and TTL percentiles. Caches sorted by IP address, and grouped by (anonymized) IATA code. December 12-18 2013, dataset *ISPI-A*.

We acknowledge that The RTT and TTL samples might 403
 be biased by issues due to the presence of congestion in the 404
 network path to the caches. However, YouLighter’s aim is to 405
 capture these kind of events as well, and notify them to the ISP, 406
 which then will investigate further to troubleshoot the probl- 407
 em. Anyhow, the correct working point of an ISP network 408
 should be far from a regime affected by congestion, so we 409
 expect congestion events to be due to actual changes in the CDN 410
 infrastructure or in the network paths to the caches, rather than 411
 in the ISP network. 412

A. Multi-Dimensional Clustering 413

We leverage above intuition to design a clustering algorithm 414
 to automatically find homogeneous groups of caches. We use 415
 some ingenuity to characterize the path from client to each 416
 cache, and then to cluster caches that exhibits similar paths. 417
 We can split the process of our methodology into the following 418
 steps: 419

Step 1 - Passive monitoring of YouTube video flows: As 420
 described in Sec. III, a passive probe provides the continuous 421
 collection of YouTube traffic logs. We log the metadata of each 422
 TCP connection, and we store logs in a database for further 423
 processing. 424

Step 2 - Measurement consolidation and filtering: To ease 425
 the monitoring procedure, we use a batch processing approach 426
 that considers time windows of size ΔT . Thus, every ΔT we 427
 generate a “snapshot”, and we aggregate and process measure- 428
 ments in it. In the following, we indicate the n -th snapshot as 429
 a superscript when needed, e.g., $a^{(n)}$ indicates the metric a at 430
 snapshot n . The choice of the time granularity at which a new 431
 snapshot is built is driven by the necessity of finding a trade off 432
 433

433 between the amount of data to consider, and the frequency at
 434 which the network administrator is expected to be capable of
 435 reacting to anomalies.

436 We identify each cache x by its IP address. We then group
 437 all flows in the same snapshot with the same server IP address
 438 to obtain a table where columns correspond to the metric (e.g.,
 439 RTT, TTL, transmitted packets, etc.), and each row corresponds
 440 to a sample, i.e., the tuple of measured values observed within
 441 a TCP flow.

442 Since we are interested in the active caches, we discard those
 443 with less than $MinFlow = 50$ samples. We define the whole
 444 measurement snapshot n as $X^{(n)}$.

445 **Step 3 - Feature selection and data normalization:** Next,
 446 we apply a feature selection driven by domain knowledge to
 447 select the set \mathcal{M} of *metrics*. In particular, as we are inter-
 448 ested in grouping caches according to the path properties, we
 449 choose $\mathcal{M} = \{RTT, TTL\}$. Then, for each cache x in the snap-
 450 shot X , and for each metric $m \in \mathcal{M}$, we generate a Cumulative
 451 Distribution Function (CDF). From the distribution, we extract
 452 the vector $P_m(x) = (p_{m,1}(x), p_{m,2}(x), \dots, p_{m,k}(x))$ contain-
 453 ing k percentiles of m for cache x . Then, we translate the
 454 percentile values obtained from the real space to an hypercube
 455 space of unitary size. This is a standard approach which allows
 456 the data analytics algorithm to work with a fixed parameter con-
 457 figuration. We thus standardize percentiles following a simple
 458 normalization:

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

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

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

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

461 At last, $\bar{P}_m(x) = (\bar{p}_{m,1}(x), \bar{p}_{m,2}(x), \dots, \bar{p}_{m,k}(x))$ repre-
 462 sents the standardized vector of *features* for the metric m for
 463 server x . Recalling that $\mathcal{M} = \{RTT, TTL\}$, we identify each
 464 cache $x \in X$ with a $2k$ -dimensional which is then normalized
 465 in a space of edge 1 and features:

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

466 and we transform the original set of caches X into a set of points
 467 $\bar{X} = \{\bar{x}\}$.

468 **Step 4 - Clustering:** Among different clustering algorithms
 469 we employ the DBSCAN (Density-Based Spatial Clustering
 470 of Applications with Noise) algorithm [6] to group together
 471 caches based on their multi-dimensional features. We choose
 472 DBSCAN because (i) it is able to handle clusters of arbitrary
 473 shapes and sizes; (ii) it is relatively resistant to noise and out-
 474 liers; (iii) it does not require the specification of the number
 475 of desired clusters (iv) it is a density based clustering algorithm
 476 which perfectly fits our scenario. DBSCAN is a clustering algo-
 477 rithm presented in 1996 [6] which exploits the notion of dense
 478 area to clusterize elements. This characteristic is particularly
 479 useful in our scenario since caches belonging to the same edge-
 480 node should own similar metrics value lying in a small region
 481 of the possible space. Even in presence of small difference

within the same edge-node, DBSCAN has good performance 482
 as it can handle clusters of different shape. Another important 483
 point is that DBSCAN requires only two parameters: ϵ and 484
 $MinPts$. ϵ determines the maximum allowed distance between 485
 any given point in a cluster and its closest neighbour belong- 486
 ing to the same cluster, and $MinPts$ the minimum number 487
 of points required to form a cluster. Based on that, it classifi- 488
 es all points as being (i) core points, i.e., in the interior of 489
 a dense region; (ii) border points, i.e., on the edge of a dense 490
 region; or (iii) noise points, i.e., in a sparsely occupied region. 491
 Noise points do not form any cluster, while the algorithm puts 492
 in the same cluster any two core points that are within ϵ of each 493
 other. Similarly, any border point that is close enough to a core 494
 point is put in the same cluster as the core point. The result of 495
 this process is a collection \mathcal{C} of clusters $C_j \in \mathcal{C}$, also named as 496
clustering: 497

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

B. Highlighting Changes With the Pattern Dissimilarity 498

We are now interested in tracking the evolution of clusters 499
 over time, for which, we adapt the methodology presented in 500
 [16], as we discuss in Sec. I. As authors presented in [16] 501
 compare two clusterings $\mathcal{C}1$ and $\mathcal{C}2$ obtained considering two 502
different datasets, i.e., snapshots in our case opens many differ- 503
 ent scenarios. For instance, i) points that were present in $\mathcal{C}1$ may 504
 not be present in $\mathcal{C}2$, and vice versa; ii) points clustered into the 505
 same cluster in $\mathcal{C}1$ can now belong to two or more clusters in 506
 $\mathcal{C}2$; and iii) the same points that form a cluster in $\mathcal{C}1$ can still 507
 form the same cluster, but can be placed in another region in the 508
 clustering space in $\mathcal{C}2$. In our case, this corresponds to i) popu- 509
 lar caches at snapshot n that are not anymore used at snapshot 510
 $n + 1$, or ii) some caches at snapshot n that were part of the 511
 noise are instead clustered at snapshot $n + 1$, or iii) the path to 512
 caches suddenly changes at snapshot $n + 1$, altering RTT and 513
 TTL. Since we do not have any mean to label the clusters in two 514
 different clustering results for instance by using a ground truth 515
 label. We can not evaluate major changes, i.e., the presence of a 516
 totally new cluster by simply evaluating the difference of cluster- 517
 ing results by comparing which clusters are shared between 518
 the snapshots. Therefore, to evaluate the difference between the 519
 clustering, we adapt the historic cost function presented in [16], 520
 and we introduce the notion of *Pattern Dissimilarity*. 521

1) *Clustering Patterns:* We first map each cluster into a sin- 522
 gle *Centroid* that summarizes it. For each cluster, a centroid is 523
 computed following the standard approach, i.e., it is the mean 524
 position of all the points in all of the coordinate directions. 525
 Therefore, given a cluster $C \in \mathcal{C}$, we consider the centroid, or 526
 geometric center, \hat{x} whose components $\hat{p}_{m,i}$ in the i percentile 527
 of feature m are: 528

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

All centroids then form a *pattern* $\hat{\mathcal{P}} = \{\hat{x}\}$. The *renorm()* func- 529
 tion considers the re-normalization of features that is needed 530
 if points in $\mathcal{C}1$ and $\mathcal{C}2$ went through different normalization 531

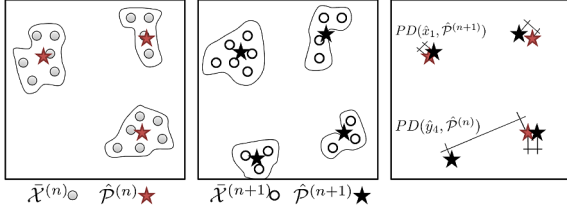


Fig. 6. Example of Clusterings, Patterns and Centroid Distance computations.

processes according to Eq. (3). In other words, $renorm()$ transforms the space defined by the centroids obtained by two different snapshots in a common hypercube space of unitary side. This normalization step is required since normalizing over all snapshots is not possible because in an online analysis system future snapshot are not known a priori. 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:

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

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

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

To better explain the need of this two-step normalization process we describe a simple one-dimensional example. Consider the case in which the first snapshot has values varying in $[0, 15]$, and the second snapshot in $[2, 22]$. We first perform a per-snapshot normalization to run DBSCAN, so that we obtain $[0, 15] \rightarrow [0, 1]$ as by Eq. (3). Similarly $[2, 22] \rightarrow [0, 1]$ for the second snapshot. Next, to compare these two snapshots, the $renorm()$ function will re-normalize the space in a common space with $[0, 22] \rightarrow [0, 1]$.

2) *Centroid Distance*: Given a centroid \hat{x} and a centroid pattern \hat{P} , we define the *Centroid Distance (CD)* as the distance between \hat{x} and its closest centroid in \hat{P} . Specifically, we compute the closest centroid $\hat{y}^* \in \hat{P}$ such that $d(\hat{x}, \hat{y}^*) \leq d(\hat{x}, \hat{y}) \forall \hat{y} \in \hat{P}$. $d(x, y)$ 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{P} is

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

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 $\hat{P}1$ and $\hat{P}2$ may be different, we need to symmetrize the definition:

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

Fig. 6 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 $\hat{P}^{(n)}$ and $\hat{P}^{(n+1)}$, and finally compute their dissimilarity $PD(\hat{P}^{(n)}, \hat{P}^{(n+1)})$.

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 input data. We consider two main sources of changes: i) centroids that simply move from their position, and ii) the birth of new centroids reflecting the generation of a new cluster in the data.

1. *Changes within the Same Metric Space*: Consider first that case where changes to samples leave samples within the same metric space, i.e., $min_m^{(n)} = min_m^{(n+1)}$ and $max_m^{(n)} = max_m^{(n+1)}$.

For the first scenario, we generate a random pattern $\hat{C}1$ of $N = |\hat{C}1|$ 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 $\hat{C}2$ by taking the centroids in $\hat{C}1$, and repositioning them in a random sphere of radius e centered in the centroid original position. Finally, we compute $PD(\hat{C}1, \hat{C}2)$. We repeat the experiment for 100 times, and compute the average and standard deviation of the obtained values. Fig. 7(a) reports the average *Pattern Dissimilarity* and its standard deviation 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* and its standard deviation.

For the second case, we run the same experiment while also increasing the number of centroids. Thus $|\hat{C}1| < |\hat{C}2|$. Fig. 7(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 i.e., edge-nodes in our scenario.

2. *Changes to a Larger Metric Space*: Consider now the that case where changes move part of the samples outside the

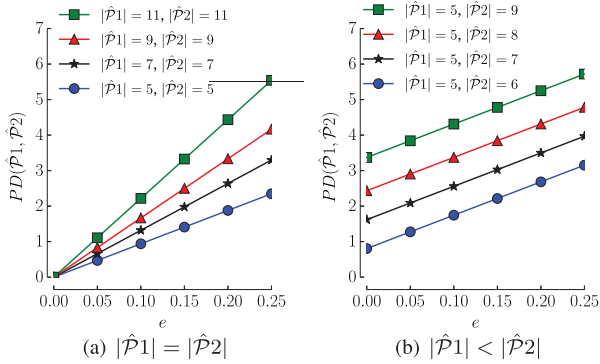


Fig. 7. *Pattern Dissimilarity* for increasing noise e , for constant and increasing number of centroids.

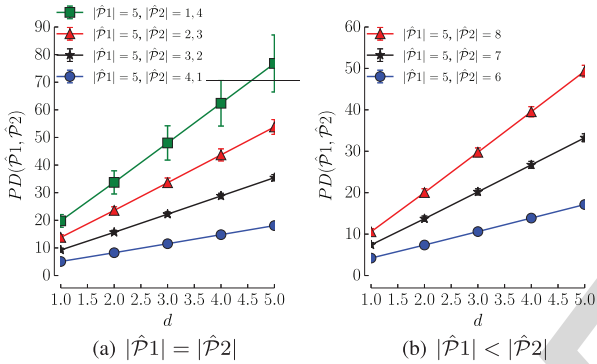


Fig. 8. *Pattern Dissimilarity* for increasing distance among clusters d , for constant and increasing number of centroids.

metric space at time n . In particular, let samples move/appear with $\max_m^{(n+1)} = d \max_m^{(n)}$, $d > 1$.

Fig. 8 shows results. Starting with left plot, we consider the case where 5 clusters are present at both time. At time $n + 1$, points of k clusters move in a different portion of the space, being allowed to move in a space of diameter d bigger than the original one. Results show that this change has a much greater impact, see Fig. 8(a). In the figure, $|\hat{P}_2| = x, y$ states that x cluster still lie in the original space, and y lie in the upscaled space. Intuitively, since points now move to a larger space, the *Pattern Dissimilarity* of the cluster which lies outside the original space let the *Pattern Dissimilarity* grow higher.

Fig. 8(b) consider the birth of new clusters, which lie in the upscaled space of factor d . Also in this case, the *Pattern Dissimilarity* grows much higher than when considering the birth of clusters within the same space – cfr. Fig 7(b).

These simple experiment allows us also to select thresholds to highlight important changes, rather than minor changes. For instance, when samples move outside the original space, the *Pattern Dissimilarity* is typically larger than 10. Conversely, when changes move samples within the same space, *Pattern Dissimilarity* is smaller than 10. This will be useful when highlighting significant changes in real data.

D. Complexity

After evaluating different aspects of the methodology, the last important part to analyze is its complexity. This step is required to understand the feasibility of running *YouLighter* in a real

environment. To evaluate *YouLighter* complexity we study the computational cost of the different steps. We assume that the network has been instrumented to extract flow level logs from passive observations of packets. In our work we rely on Tstat a tool developed at Politecnico di Torino able to perform live traffic monitoring up to few Gb/s using off-the-shelf hardware [23], [27].

Assume to have N caches and a total number of YouTube flows F . From our dataset, we always obtain N in the order of 1,000 and F amounting to half billion flows in each week of data from ISP1. Therefore, a one week long dataset can be easily stored in memory of MB magnitude. Given the flow level log, the first step consists in measurement consolidation. It consists in grouping flows by cache and filter less used ones. Using a hash-based data structure where the key is the cache IP address, counting flows per cache has a complexity of $O(F)$.

The second step consists in feature extraction and data normalization for each flow. For each cache, empirical percentiles must be computed which entails sorting of flows. Worst case complexity is $O(F \ln(F))$ when $N = 1$ and all flows belong to the same cache. Assuming instead that F flows are equally split among N caches, the complexity becomes $O(N \frac{F}{N} \ln(\frac{F}{N})) = O(F \ln(\frac{F}{N}))$.

The third step is running DBSCAN. According to authors in [6], the algorithm complexity is $O(N \ln(N))$. The output consists in M clusters. The final step consists in computing the *Pattern Dissimilarity*. Firstly, it requires to compute the boundary values for the *renorm* function. This operation is linear $O(N)$ since it requires to find the minimum and maximum values of features. Secondly, it computes the *centroid* of each pattern. For each the M patterns, each containing L caches, we have a complexity of order $O(ML)$. Assuming L to be $O(\frac{N}{M})$, the complexity reduces to $O(N)$. The next part has to deal with the centroid distance. This part has a computational complexity of $O(M^2)$. Finally the *Pattern Dissimilarity* is computed with a cost of $O(M)$ since we have to sum all pattern distances. Overall, *YouLighter* complexity is thus dominated by the second step, i.e., $O(F \ln(\frac{F}{N}))$. For our *YouLighter* implementation, we used *Python* program language which took on average less than two minutes to perform all computation for one week of data.

V. DBSCAN PERFORMANCE

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. Clustering Performance Metrics

We first evaluate the impact of the parameter settings on the DBSCAN 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

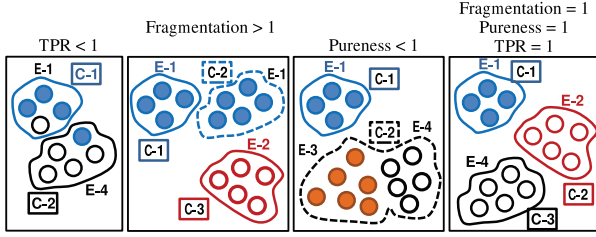


Fig. 9. Examples of patterns for which the *True Positive Rate*, the *Fragmentation Index*, and the *Purity Index* are not equal to 1, and the optimal case in which they are all equal to 1. Color represent the GT-label.

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.

To validate the clustering we obtain with DBSCAN we consider three different indices, as follows:

$$TPR = \frac{N_{TP}}{|X|}, \mu = \frac{N_C}{N_L}, \phi = \frac{N_L}{N_{GT}} \quad (12)$$

- 1) The True Positive Rate ($TPR \leq 1$), also known in the literature as ‘‘Purity’’ [26] measures 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) mislabeled caches (or FP), or ii) noise points (unlabeled points). Leftmost sub-figure in Fig. 9 reports a simple example where the clustering algorithm mislabels a cache for both the GT-labels E-1 and E-4, thus leading to $TPR < 1$. Colors represent the GT-label.
- 2) The *Fragmentation Index* ($\mu \geq 1$) is a custom metric that 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. 9 reports an example where the clustering algorithm splits edge-node E-1 in two different clusters, C-1 and C-2, thus leading to $\mu > 1$.
- 3) *Purity Index* ($\phi \leq 1$) is also a custom metric that 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. 9 reports an example where the clustering algorithm groups together edge-nodes with GT-labels E-3 and E-4 in cluster C-2, thus leading to $\phi < 1$.

Rightmost sub-figure in Fig. 9 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.

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

Fig. 10(a) 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.

Fig. 10(b) depicts results for varying ϵ . Unfortunately, DBSCAN shows a good clustering for a tiny interval of values of ϵ , e.g., $\epsilon = 0.035$. For $\epsilon > 0.035$, DBSCAN merges edge-nodes together, so that $\mu > 1$ and $\phi < 1$. By investigating further, we observe that the mean and standard deviation vary widely among caches in the same edge-node. This variability

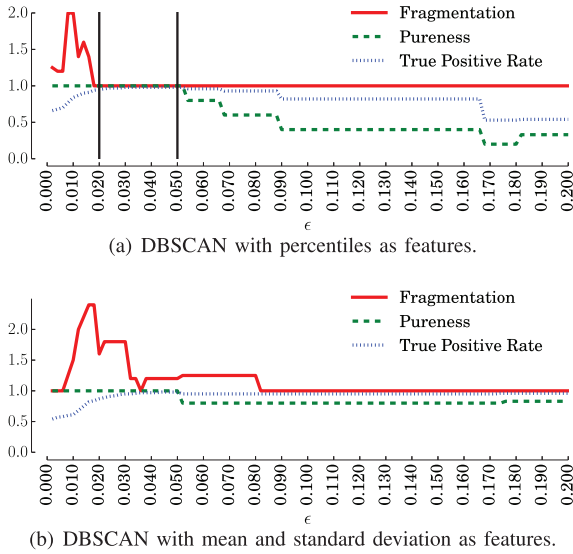


Fig. 10. DBSCAN with different feature settings. Performance versus ϵ . 1st week of November, *ISP1 - A*.

is due to the tails of the distributions which include outliers, e.g., very large RTT samples which bias the mean and standard deviation, but have little or no impact on the percentiles. Indeed, the percentiles of caches in the same edge-node are very similar, except those that gauge the tail (see the 95th percentiles in Fig. 5). 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 observe no significant differences if we avoid 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

In this section 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. The choice of the time granularity is driven by the nature of the anomalies that the ISP would like to highlight. Indeed, with too small time scales, e.g., order of hours, more alarms would be possibly raised due to the natural control system of CDNs. At the same time, an anomaly that lasts order of hours would possibly be not relevant for end-users, since the system would return to normality in short time. Snapshots form a sliding window that moves forward every day, and aggregates statistics for the past seven days. $\Delta T = 7$ days guarantees us to obtain a statistically significant amount of feature measurements for each of the caches in the set responsible of carrying the 90% of traffic.

Fig. 11 shows the evolution of the *Pattern Dissimilarity* (red solid curve, left y-axis) over time. It also depicts the evolution over time of the number of caches that remain in the noise after clustering (black dashed curve, right y-axis). From left to right,

plots refer to *ISP1-B*, *ISP1-A* and *ISP1-C*. X-axis reports daily snapshots, starting from April 1st, 2013.⁷

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 three different datasets. It starts on May 2nd (snapshot 27), May 7th (snapshot 32), and May 13th (snapshot 38) for *ISP1-B*, *ISP1-A* and *ISP1-C*, 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, top plot of Fig. 12 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), *YouLighter* identifies two clusters since the path to reach caches differs. This is clearly shown by the RTT percentiles, which makes them like to be in two different locations, with the former possibly being severely congested. We call these new cluster Bad-E-2, in opposition to caches showing small RTT, i.e., Good-E-2. While RTT distribution clearly allows us to identify a sudden CDN change, the TTL measurements in the bottom plot of Fig. 12 does not reveal any additional information with respect to what RTT already does.

We now analyze the impact of such change on the Quality of Experience the ISP customers perceive. We report in Fig. 13 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

⁷PoP referring to *ISP1-C* suffered an outage from mid July 2013 to the end of September.

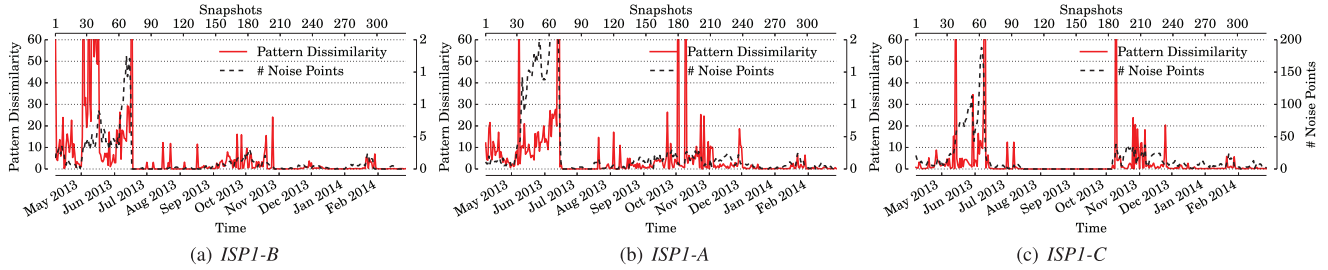
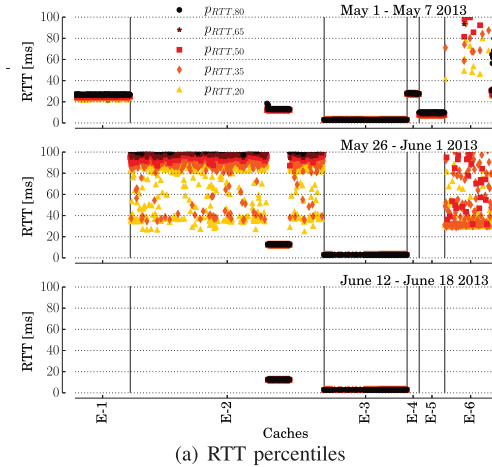
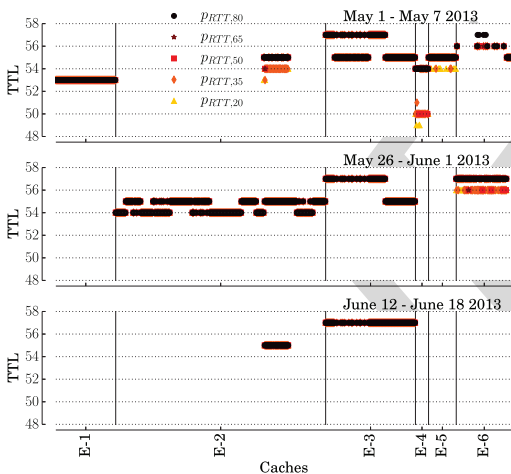


Fig. 11. Pattern Dissimilarity values and number of noise points for different traces from ISP1.



(a) RTT percentiles



(b) TTL percentiles

Fig. 12. Per-cache RTT and TTL percentiles during the ISP-wide anomaly in May 2013. Dataset *ISP1-A*.

892 QoE [28]. Tab. II corroborates above observation reporting the
 893 fractions of video (and audio) formats seen in flows handled
 894 by both Good-E-2 and Bad-E-2.⁸ For this analysis we consider
 895 only DASH formats, as for these formats the cache delivering
 896 the video automatically adapts the quality of the video stream
 897 depending on the congestion it measures on the path to the
 898 client. As shown, Good-E-2 serves larger fractions of high-
 899 definition videos. Conversely, the share of videos encoded with

⁸Observe that in our dataset only a tiny portion ($\sim 1\%$) 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.

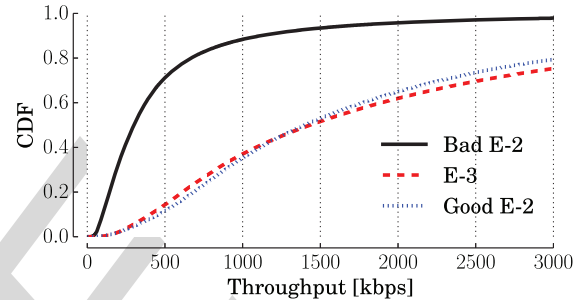
Fig. 13. Throughput distribution for flows served by E-3, Good-E-2 and Bad-E-2 during the large anomaly we observe in May 2013. Dataset *ISP1-B*.

TABLE II
 FRACTIONS OF VIDEO AND AUDIO DASH FORMATS SERVED BY
 GOOD-E-2 AND BAD-E-2. 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%

low-definition (144p and 240p) increases for Bad-E-2. This
 900 confirms that Bad-E-2 experienced possible congestion during
 901 the monitored period, severely impairing the QoE of the users.
 902

By double checking this event with the ISP network support
 903 team, we confirm the incident involved most of their customers,
 904 increasing dramatically the complaining at their customer sup-
 905 port. This confirms the pervasiveness of this event upon ISP
 906 customers.
 907

The accident reported above is an example which testifies
 908 that changes in the CDN may raise issues in the video delivery,
 909 finally harming users' experience. This highlights how impor-
 910 tant for the ISP is to monitor and pinpoint changes in the
 911 YouTube CDN. The task of measuring how the variability of
 912 the CDN structure may impact on the QoE perceived by the
 913 users is beyond the focus of this paper.
 914

B. Other Events for ISP1

We manually cross check other events, and find that some
 916 of those affected only part of the ISP customers. This shows
 917 that YouTube CDN allocates customers to edge-nodes using a
 918

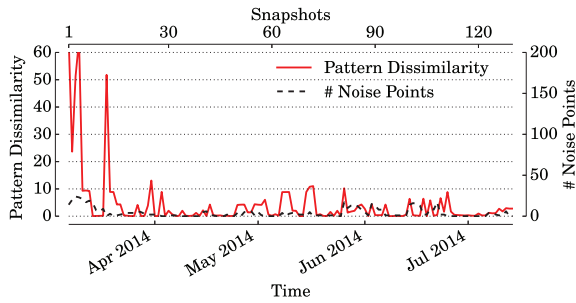


Fig. 14. *Pattern Dissimilarity* values and number of noise points for dataset *ISP2*.

919 fine grained granularity, i.e., the load-balancing allows to identify
 920 small groups of clients by using the client IP address (or
 921 network). For instance, on October 2nd (snapshot 180) and
 922 October 9th (snapshot 187) *YouLighter* highlights two sudden
 923 changes in the *ISP1-A* and *ISP1-C*, as the *Pattern Dissimilarity*
 924 peaks over 60. Inspecting the astral distances one by one, we
 925 observe that the changes are due to 3 edge-nodes (E-4, E-5 and
 926 E-6) out of 7 that suddenly “appear” in snapshot 180 and “dis-
 927 appear” in snapshot 187. The remaining four edge-nodes then
 928 serve the videos for customers in *ISP1-A* and *ISP1-C*. We analyze
 929 the impact of the presence of such caches on the QoE by measuring
 930 the aggregate download throughput before, during and after their
 931 permanence, but we do not appreciate any significant change.
 932 Also in this case we double check the event with the ISP support
 933 team and we confirm that the change had no influence on the QoE
 934 as the customer support did not receive any meaningful complain-
 935 ing in the considered period.

936 C. Events in *ISP2*

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

944 We repeat the experiment of Fig. 11 for *ISP2* dataset, and we
 945 analyze the evolution of the *Pattern Dissimilarity* and number
 946 of noise points. We report the results in Fig. 14. To check if the
 947 clustering correctly identifies the edge-nodes, we select five dif-
 948 ferent snapshots at random among the ones where *YouLighter*
 949 highlights no events. Again, we use the IATA codes as ground
 950 truth, and we manually check IP address subnets, RTTs and
 951 TTLs to see if some suspicious cache is present in a cluster.
 952 The clustering results in perfect match with the (possible) edge-
 953 nodes in the ground truth. This despite edge-nodes, path, and
 954 ISP in this dataset are completely different.

955 We then check two suspicious events. The first one occurs
 956 from March 7th to March 10th, 2014 (snapshots 1-4, $PD >$
 957 60), and the second one happens on March 18th, 2014 (snap-
 958 shot 12, $PD >$ 50). We observe that the first anomaly is due
 959 to a change in the network path to reach a small group of
 960 caches in E-2. We observe that this deviation does not influ-
 961 ence the QoE perceived by the users. For the second event,
 962 by comparing the clustering at snapshot 12 with the following

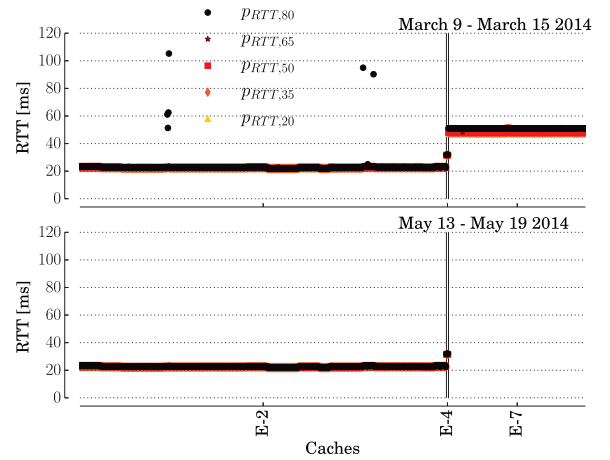


Fig. 15. Per-cache RTT percentiles during the second *ISP-wide* anomaly in March 2014. Dataset *ISP2*.

963 snapshot, i.e., snapshot 13 (March 19th), we observe a notable
 964 change in the infrastructure of the YouTube CDN: as depicted
 965 in Fig. 15 which compares the per-cache RTT percentiles, all
 966 caches belonging to edge-node E-7 disappear. Also in this case,
 967 the change has no evident impact on users’ QoE, as the average
 968 download throughput does not vary. However, we notice
 969 that the edge-node E-7 represents a much more expensive route
 970 for the *ISP2*, since it is located in an remote ISP for which no
 971 peering agreements are in place.

972 VII. COUNTERMEASURES

973 In this section, we present some possible techniques that ISPs
 974 may consider to this end, leaving a thorough design and analysis
 975 for future work.

976 Firstly, ISPs position allows them to partially control the
 977 network routing of traffic. Thus, ISPs may employ traffic
 978 engineering techniques to control YouTube traffic using both
 979 intra- and inter-routing algorithm. By using BGP, ISPs may
 980 reduce congestion by redistributing traffic among different
 981 peering links [29]. By announcing via BGP reachability of
 982 their network through other Autonomous System (AS) the
 983 ISP can influence YouTube cache allocation policy as well.
 984 For instance, *ISP1* used this technique to mitigate problems
 985 highlighted in Fig. 13. In that specific case, *ISP1* forced its AS
 986 number to be seen as reachable through another ISP (*ISP2*),
 987 located in a different country. This caused YouTube CDN
 988 to change the cache allocation policy so that customers from the
 989 *ISP1* were directed to a edge-node located in the *ISP2* country.
 990 YouTube traffic was then coming from *ISP2* to *ISP1*, through
 991 a high-capacity and uncongested peering link, de facto solving
 992 the congestion problem.

993 A second and more controversial solution would be using
 994 DNS directly. By enforcing policies on its DNS servers, the
 995 ISP may force a video request to be served by a specific edge-
 996 node or even by a specific cache, overruling the decision of the
 997 YouTube CDN. As explained in Sec. III-B, as soon as a client
 998 requests a video, YouTube selects a cache and returns the cache
 999 hostname to the client e.g. r7—fra07t16.c.youtube.com.
 1000 After receiving this hostname the client sends a DNS query
 1001 to the DNS server (typically managed by the ISP) to retrieve

the IP address of the hostname. As a consequence, in case the ISP knows that `r7---fra07t16.c.youtube.com` performance are poor, the ISP DNS can return the IP address of a different cache. This is possible since according to [3], [4], any cache can serve any video. This solution, allows the ISP to select which cache has to manage each request. Therefore the ISP may potentially override YouTube load balancing policies. Some drawbacks have to be faced. First, even if previous studies discovered that any cache can serve any video, the ISP can not be sure that its choice will not introduce a bigger latency due to cache miss, that can cause a further redirection to other caches. Second, the ISP can not control whether a cache will be switched off for maintenance. Observe that an ISP attempt to react to changes in the CDN infrastructure could cause further reactions on YouTube's side too. As such, careful investigations must be done to design and study this kind of policies.

VIII. CONCLUSION

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* is able to automatically group thousands of caches into few edge-nodes. To then compare the results of clustering obtained considering different snapshots collected in consecutive time intervals, we propose the *Pattern Dissimilarity* which allows to easily pinpoint changes in clusters.

YouLighter is validated using a large dataset of traces reporting the activity of users regularly accessing YouTube. Our results are excellent: we show that after a short and simple tuning procedure to find the best setup for DBSCAN, *YouLighter* can detect anomalous events that happened in YouTube CDN. For instance, we could notice a large transformation in a crucial edge-node of YouTube CDN which notably impaired the QoE perceived by the monitored 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 *Proc. ACM Internet Meas. Conf. (IMC)*, 2013.
 [2] V. Adhikari, S. Jain, Y. Chen, and Z.-L. Zhang, "Vivisectioning YouTube: An active measurement study," in *Proc. IEEE INFOCOM*, 2012, pp. 2521–2525.

[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 *Proc. IEEE 31st Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2011, pp. 248–257.
 [4] P. Casas, P. Fiadino, and A. Bär, "Understanding HTTP traffic and CDN behavior from the eyes of a mobile ISP," in *Proc. Passive Active Meas. (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," *IEEE Trans. Netw. Serv. Manage.*, 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 *Proc. ACM Int. Conf. Knowl. Discovery Data Min. (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*. New York, NY, USA: 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," *Comput. Netw.*, vol. 51, no. 12, pp. 3448–3470, Aug. 2007.
 [9] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," in *Proc. Comput. Surveys*, 2009, vol. 41, pp. 1–58.
 [10] H. Yan *et al.*, "Argus: End-to-end service anomaly detection and localization from an ISP's point of view," in *Proc. IEEE INFOCOM*, 2012, pp. 2756–2760.
 [11] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 217–228, Aug. 2005.
 [12] J. Jiang, V. Sekar, I. Stoica, and H. Zhang, "Shedding light on the structure of Internet video quality problems in the wild," in *Proc. 9th ACM Conf. Emerging Netw. Exp. Technol. (CoNEXT'13)*, 2013, pp. 357–368.
 [13] P. Fiadino, A. D'Alconzo, A. Bar, A. Finamore, and P. Casas, "On the detection of network traffic anomalies in content delivery network services," in *Proc. 26th Int. Teletraffic Congr. (ITC)*, 2014, pp. 1–9.
 [14] G. Münz, S. Li, and G. Carle, "Traffic anomaly detection using K-means clustering," in *Proc. GIITG Workshop MMBnet*, 2007.
 [15] R. D. Torres, M. Y. Hajjat, S. G. Rao, M. Mellia, and M. M. Munafo, "Inferring undesirable behavior from P2P traffic analysis," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, pp. 25–36, 2009.
 [16] D. Chakrabarti, R. Kumar, and A. Tomkins, "Evolutionary clustering," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Min. (KDD)*, 2006, pp. 554–560.
 [17] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *Proc. 29th Int. Conf. Very Large Data Bases (VLDB'03)*, 2003, pp. 81–92.
 [18] M. K. Goldberg, M. Hayvanovych, and M. Magdon-Ismael, "Measuring similarity between sets of overlapping clusters," in *Proc. IEEE SocialCom*, 2010, pp. 303–308.
 [19] P. Kalnis, N. Mamoulis, and S. Bakiras, "On discovering moving clusters in spatio-temporal data," in *Advances in Spatial and Temporal Databases*. New York, NY, USA: Springer, 2005, vol. 37, pp. 364–381.
 [20] Z. Li, B. Ding, J. Han, and R. Kays, "Swarm: Mining relaxed temporal moving object clusters," in *Proc. VLDB Endowment*, 2010, vol. 3, pp. 723–734.
 [21] D. Kifer, S. Ben-David, and J. Gehrke, "Detecting change in data streams," in *Proc. 30th Int. Conf. Very Large Data Bases (VLDB)*, 2004.
 [22] D. Giordano *et al.*, "YouLighter: An unsupervised methodology to unveil YouTube CDN changes," in *Proc. 27th Int. Teletraffic Congr.*, 2015.
 [23] A. Finamore, M. Mellia, M. Meo, M. M. Munafo, and D. Rossi, "Experiences of Internet traffic monitoring with Tstat," *IEEE Netw.*, vol. 25, no. 3, pp. 8–14, May/June. 2011.
 [24] D. Naylor *et al.*, "The cost of the S in HTTPS," in *Proc. 10th ACM Int. Conf. Emerging Netw. Exp. Technol.*, 2014, pp. 133–140.
 [25] 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 *Proc. Internet Meas. Conf. (IMC)*, 2012.
 [26] P.-N. Tan *et al.*, *Introduction to Data Mining*. Reading, MA, USA: Addison-Wesley, 2006, vol. 1.
 [27] D. Rossi and M. Mellia, "Real-time TCP/IP analysis with common hardware," in *Proc. IEEE Int. Conf. Commun.*, 2006, vol. 2, pp. 729–735.
 [28] P. Casas, A. Sackl, S. Egger, and R. Schatz, "YouTube AMP; Facebook quality of experience in mobile broadband networks," in *Proc. IEEE Globecom Workshops*, Dec. 2012, pp. 1269–1274.
 [29] B. Quoitin, C. Pelsser, L. Swinnen, O. Bonaventure, and S. Uhlig, "Interdomain traffic engineering with BGP," *IEEE Commun. Mag.*, vol. 41, no. 5, pp. 122–128, May 2003.

1134
1135
1136
1137
1138
1139
1140
1141
1142
1143



Danilo Giordano received the M.Sc. degree in computer engineering from Politecnico di Torino, Torino, Italy, in 2013. He is currently pursuing the Ph.D. degree at the Telecommunication Networks Group, Politecnico di Torino. In Summer 2014, he was a Research Intern with Narus Inc., now part of Symantec, working on anomaly detection techniques. His research interests include network traffic characterization, monitoring, anomaly detection, and security on the Internet.

1144
1145
1146
1147
1148
1149
1150
1151
1152



Stefano Traverso (M'12) has been visiting Telefonica I+D Research Center (Barcelona, Spain), NEC Laboratories (Heidelberg, Germany), and Alcatel-lucent Bell Labs (Paris, France) during his Ph.D. and postdoctoral studies. His research interests include privacy-preserving systems, network measurements and content delivery networks. He is currently a Postdoc Fellow of the Telecommunication Networks Group, Politecnico di Torino.

1153
1154
1155
1156
1157
1158
1159
1160
1161



Luigi Grimaudo received the M.Sc. and Ph.D. degree in information and system engineering from the Computer Engineering Department, Politecnico di Torino, Torino, Italy, in 2010 and 2013, respectively. From 2011 to 2014, he collaborated with Narus Inc. working on traffic classification problems and online social networks analysis. His research interests include Internet traffic classification, recommendation system, social network analysis, and big data.

1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172



Marco Mellia (SM'09) received the Ph.D. degree in electronic and telecommunication engineering from the Politecnico di Torino, Torino, Italy, in 2001. He held a position as Associate Professor with Politecnico di Torino. In 2002, he visited the Sprint Advanced Technology Laboratories working at the IP Monitoring Project (IPMON). From 2011 to 2013, he collaborated with Narus Inc., CA, USA, working on traffic monitoring and cyber-security system design. He has coauthored over 250 papers published in international journals and presented in leading international conferences, all of them in the area of telecommunication networks.

1173 His research interest include traffic monitoring, and big data analysis. He participated in the program committees of several conferences including the ACM SIGCOMM, the ACM CoNEXT, the ACM IMC, the IEEE Infocom, the IEEE GLOBECOM, and the IEEE ICC. He is an Area Editor of ACM *Computer Communication Review*, the ACM/IEEE TRANSACTIONS ON NETWORKING, and the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT. He is currently the co-ordinator of the mPlane Integrated Project that focuses on building an intelligent measurement plane for future network and application management.



Elena Baralis (M'99) received the M.Sc. degree in electrical engineering and the Ph.D. degree in computer and systems engineering from Politecnico di Torino, Torino, Italy. She has been holding a position as Full Professor with the Control and Computer Engineering Department since 2005. Her research interests include database systems, data mining, algorithms for diverse data mining tasks in a Big Data environment, and on different domains where data mining techniques find their application. She has authored over 100 papers in international peer-reviewed journals and conference proceedings.

1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194



Alok Tongaonkar (SM'14) is a Data Scientist Director leading the Center for Advanced Data Analytics (CADA) team at Symantec Corporation. His research focuses on network security and management. Prior to joining Symantec, Alok was a Principal Member of Technical Staff in the CTOs Office at Narus, Inc. where he led the research and development of many novel technologies in the area of network protocol reverse-engineering and pioneered automated mobile app identification techniques. He has served as a reviewer for many peer-reviewed publications such as the IEEE TRANSACTIONS ON INFORMATION FORENSICS JOURNAL and the *IEEE Internet Computing Magazine*.

1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207



Sabyasachi Saha received the Ph.D. degree in computer science from the University of Tulsa, Tulsa, OK, USA. He is a Principal Data Scientist with Cyphort Inc., Santa Clara, CA, USA. Previously, he was with Symantec, Narus Inc., and Toyota ITC. His research interests include machine learning, data mining, artificial intelligence, and network security.

1208
1209
1210
1211
1212
1213
1214