SeLINA: a Self-Learning Insightful Network Analyzer

SeLINA: a Self-Learning Insightful Network Analyzer

Daniele Apiletti, Elena Baralis, *Member, IEEE,* Tania Cerquitelli, *Member, IEEE,* Paolo Garza, *Member, IEEE,* Danilo Giordano, Marco Mellia, *Senior Member, IEEE,* and Luca Venturini

***Abstract*—Understanding the behavior of a network from a large scale traffic dataset is a challenging problem. Big data frameworks offer scalable algorithms to extract information from raw data, but often require a sophisticated fine-tuning and a detailed knowledge of machine learning algorithms. To streamline this process, we propose SeLINA (Self-Learning Insightful Network Analyzer), a self-tuning tool to extract knowledge from network traffic measurements. SeLINA includes different data analytics techniques providing self-learning capabilities to state-of-the-art scalable approaches, jointly with parameter auto-selection to off-load the network expert from tuning. We combine both unsupervised and supervised approaches to mine data with a scalable approach. SeLINA embeds mechanisms to check if the new data fits the model, to detect possible changes in the traffic, and to, possibly automatically, trigger model rebuilding.**

The result is a system that offers human-readable models of the data with minimal user intervention, supporting domain experts in extracting actionable knowledge and highlighting possibly meaningful interpretations. SeLINA's current implementation runs on Apache Spark. We tested it on large collections of real-world passive network measurements from a nationwide ISP, investigating YouTube and P2P traffic. The experimental results confirmed the ability of SeLINA to provide insights and detect changes in the data that suggest further analyses.

***Index Terms*—Mining and statistical methods; Machine learning; Network data analysis**

I. Introduction

Internet monitoring has always played a fundamental role in understanding how the network is performing, how users are accessing resources, and how to properly control and manage the infrastructure. The growth of traffic, users, services and applications running in the internet challenges everyday the network administrators and analysts to cope with system complexity and in the understanding of how it works. Big data and machine learning approaches have emerged to build systems that aim at automatically extracting information from the raw data that the monitoring infrastructures offer, and a significant effort has been devoted to apply them to network traffic analysis. Most of the proposed systems target a specific problem, e.g., monitoring of a CDN [1], [2], [3], detecting anomalies [4], [5], or simply offering scalable platforms [6].

However, few works have targeted the general-purpose extraction of useful information from the raw data exposed by the system, i.e., the application of the data mining approach to information discovery, a classic application of unsupervised machine learning approaches. While methodologies exist, to the best of our knowledge, they require non-trivial skills and the domain expert needs to be able to fine tune the underlying algorithms. In this work, we target the design of an unsupervised machine learning tool that allows the network administrator to discover properties of the traffic, without requiring her to be a machine learning expert. We identified the following requirements.

- Scalability, as the ability to (i) process very large datasets, but (ii) provide compact representations of the traffic, independently of the data size.
- Auto-configuration, as the capability to (i) self-adapt to different data (e.g., data densities, cluster shapes), and to (ii) self-tune the algorithm parameters to avoid human intervention.
- Human-readability of both results and underlying models, to make the knowledge better exploitable and more actionable.
- Self-assessment and self-evolution, to autonomously evaluate the model quality and trigger a rebuilding when the model fitting to new data degrades.

The above-mentioned design guidelines led to the design of SeLINA (Self-Learning Insightful Network Analyzer), which exploits both supervised and unsupervised data-mining techniques by combining their strengths. Specifically, unsupervised approaches are used to autonomously identify clusters of homogeneous traffic flows, thus reducing the granularity of objects to observe from millions of single flows to few tens of clusters, and generating a model of traffic. Human-readable and fast supervised approaches are used then to classify flows on the fly and assign them to clusters, and to offer valuable information about the main characteristics of each class. The system computes internal quality indices to check whether the new data does not fit anymore the historical model, suggesting to the analyst changes in the underlying network traffic, and, possibly automatically, triggering a new clustering phase to update the model.

SeLINA has been implemented in a state-of-the-art Big Data framework, Apache Spark, and has been applied to two real-world large use cases: a YouTube video streaming dataset and a peer-to-peer traffic dataset. Experimental results show that SeLINA is able to provide insightful network traffic models, e.g., pinpoint different groups of YouTube servers with different properties, and suggest the presence of changes in the infrastructure that have caused well-known issues to end-

D. Apiletti, E. Baralis, T. Cerquitelli, P. Garza, and L. Venturini are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, Torino, Italy, email: {name.surname}@polito.it. D. Giordano and M. Mellia are with the Dipartimento di Elettronica e Telecomunicazioni, Politecnico di Torino, Corso Duca degli Abruzzi 24, Torino, Italy, email: {name.surname}@polito.it.

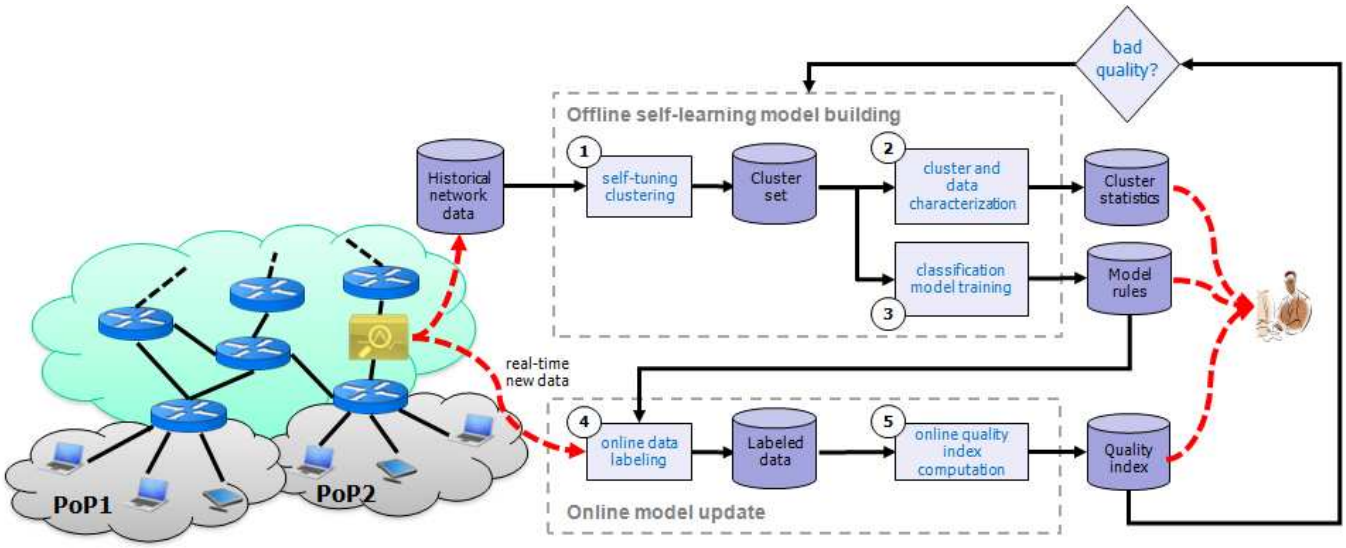


Fig. 1. SeLINA building blocks.

users [3].

This paper is organized as follows. Section II provides an overview of the proposed methodology, while Sections III-IV describe its main building blocks. Section V provides and overview of the experimental evaluation campaign, while Sections VI-VII thoroughly discuss the experiments performed on two real use cases based on real traffic datasets. Finally, Section VIII compares our approach with previous work, while Section IX draws conclusions and presents future developments of this work.

II. Methodology overview

Figure 1 depicts the main components of the proposed methodology.

Offline self-learning model building. This component, which analyzes historical network traffic flows, aims at building a self-learning data characterization model, and consists of three phases: (1) a self-tuning clustering phase, (2) a cluster and data characterization phase, and (3) a classification model training phase.

Online model update. This component analyzes new network data in real-time by applying the model built in the previous block to detect changes in the network traffic characterization. It consists of two phases: (4) a real-time data labeling phase, and (5) an online quality index computation phase.

In details, step (1) consists of a self-tuning clustering algorithm, which is run over historical data to discover homogeneous groups of flows without prior knowledge, in a fully autonomous and unsupervised fashion. Effectively applying cluster analysis on real datasets requires the non-trivial choice of algorithm-specific parameters, a typically difficult task for domain experts exploiting data mining techniques. To this aim, SeLINA includes strategies to automatically tune the clustering parameter values. In step (2), the resulting cluster set is then enriched by both general-purpose and domain-specific statistics, whose aim is to support network analysts in understanding the semantics of the identified clusters.

The cluster set is also given as input to the model training phase of step (3), where a classification model is built by exploiting clusters as classification labels. The model is able to self-learn how to assign each network flow to the proper cluster. Different classification techniques could be exploited, depending on the preference towards pure performance (e.g., accuracy) or human-readability of the model. For SeLINA we choice a decision tree algorithm, which is among the most popular classification techniques and provides an easily readable model in the form of classification rules. The latter feature supports the network analyst in getting more meaningful insights on the reasons for the classifier underlying choices.

The classification model is exploited in the real-time data labeling phase at step (4), where each observed network flow is assigned a label. Then, at step (5) the quality index computation is executed, by exploiting different quality indicators to self-assess the model fitting and its results over time. When the quality index falls below a given threshold, the offline model building can be automatically triggered to rebuild a new model better fitting the new data, thus providing self-evolutionary features.

SeLINA is a general-purpose methodology which can be easily exploited to analyze large collections of network data (e.g., network traffic headers, network flow characteristics, statistical measurements of traffic flows). As a case study, in the paper we apply SeLINA to analyze network measurements collected through Tstat [7].

III. Offline self-learning model building

The core of the SeLINA approach is the offline self-learning model building component, which consists of (1) a self-tuning clustering phase, (2) a cluster and data characterization phase, and (3) a classification model training phase. Details on each phase are provided in the following.

A. Self-tuning clustering phase

SeLINA exploits clustering to autonomously identify homogeneous groups of network traffic flows without prior knowledge. This phase performs a preliminary data normalization step, by means of the standard z-score technique [8], followed by a clustering algorithm applied to the normalized data. Among the many clustering algorithms available to this aim, SeLINA adopts an advanced DBScan-based [9] algorithm [10] providing high-quality clusters on very-large real data collections. Since the clustering phase is at the core of the SeLINA self-learning feature, in the following subsections its building blocks are presented.

1) Basic DBScan: DBScan is a density-based approach that identifies clusters as dense areas of data points surrounded by lower density spaces, whose points are marked as noise. The identification of the dense areas is driven by two parameters: $epsilon$ and $MinPoints$. Given an arbitrary point p, the density of the area of radius $epsilon$ centered in p is considered, and the points in this area are counted. If the number of points in the area is at least $MinPoints$ then p is called core point, the area is considered dense, and it is merged with adjacent dense areas to form a cluster.

DBScan is a well-known clustering algorithm, fruitfully exploited in a variety of application contexts. Its strength is the ability to identify arbitrary-shaped clusters, and isolate noise and outliers. The results provided by DBScan are usually better than those provided by other popular clustering algorithms. However DBScan requires longer execution times, due to its quadratic complexity. To scale to very large datasets, we exploit the Spark-based distributed implementation of DBScan[1] proposed by Aliaksei Litouka. The main difference with respect to the original centralized version is an additional partitioning step, performed at the beginning.

2) Self-tuning Multi-level DBScan (SMDBScan): SeLINA improves the basic DBScan approach by addressing two main issues: (i) parameter setting, and (ii) diverse data densities within the same dataset. To offload domain experts from the critical task of configuring DBScan-specific parameters, SeLINA includes a self-tuning strategy to automatically set proper values. Furthermore, very-large real datasets are often characterized by diverse data distributions in different regions, a situation hardly handled by the standard DBScan. To address both issues, the SMDBScan algorithm in SeLINA builds upon an advanced version of DBScan successfully proposed in [10]. SMDBScan features a multi-level iterative approach and a smart automatic parameter-setting procedure.

Multi-level iterative approach. At each iteration, SMDBScan considers the data points which have not been assigned to a cluster yet (at the first iteration, the whole dataset is considered). Then it (i) partitions them to allow parallel computation, (ii) automatically selects the most appropriate values of $epsilon$ and $MinPoints$, and (iii) executes the standard DBScan with such parameter settings. At the end of each iteration, the newly found clusters are included in the global set of clustering results, while the noise points become the dataset for the next iteration.

[1]Downloaded from https://github.com/alitouka/spark_dbscan

Automatic parameter setting: $epsilon$. To automatically compute the values of $epsilon$ and $MinPoints$ at each iteration, SMDBScan introduces a self-tuning procedure, consisting of two heuristics. The two heuristics are very intertwined, the second depending on the $epsilon$ set by the first, and they are designed to fit the scope of a multi-level strategy.

To determine $epsilon$, SMDBScan exploits the density-based concept of cluster: "a dense area surrounded by a lower density zone". To this aim, a greedy approach is exploited, selecting the best potential $epsilon$ for each point separately. A final decision is then taken globally given all the local best $epsilons$.

In details, given an arbitrary point p, the algorithm identifies the boundary of the dense area around p. To this aim, it computes the density distribution in the hypersphere having radius r and center in p, with increasing values of r. The larger r, the higher the number of points inside the hypersphere will be. We define dense areas when the number-of-point increasing rate is higher than the growth in volume. At the border of a dense area, this growth rate will show an inversion of the trend, as soon as the volume starts growing faster than the number of points. The proposed heuristics chooses the first inversion point as the border. If many inversion points occur, greedily choosing the first one reduces the computational time and leaves margin for further exploration in the next levels of SMDBScan. The final value of $epsilon$, actually used for each run of DBScan, is selected by considering the first quartile of the set of border values generated by applying the border-detection procedure for all points p. The first quartile value produces a set of dense clusters covering a representative subset of our data: taking the first quartile leads to having at least a quarter of all the points set as core points with high probability, which will help covering a good portion of the dataset in few levels.

The border-detection procedure increases the r value at $epsStep$ increments. This is the only parameter, whose main impact is on the execution time: very small steps lead to many iterations to converge. In our experiments, we found that a value of 10^{-3} was reasonable for the hardware at our disposal.

The pseudo-code for the $epsilon$ self-tuning is reported in Figure 2. Since the data partitions are independent, the main loop (Figure 2, lines (2)-(19)) is executed in a distributed fashion by exploiting Spark (each data partition is associated with an independent task).

Automatic parameter setting: $MinPoints$. Once $epsilon$ has been set, the value of $MinPoints$ is automatically set by selecting the value for which the product $MinPoints \times \text{numberOfCorePoints}(dataset, MinPoints, epsilon)$ is maximum. The approach stems from the following observations. $MinPoints$ represents the minimum size of the generated clusters. Small clusters are not interesting, because they represent a negligible part of our data. We are interested in the main groups and their characterization and we aim at setting high values of $MinPoints$: The higher the value of $MinPoints$, the higher the minimum cardinality of the generated clusters. However, the higher the value of $MinPoints$, the lower the number of core points will be. With lower values of $numberOfCorePoints$, the

Algorithm 1: Epsilon setting

```
Input  : Dataset partitions - dataPartitions
Input  : Epsilon step - epsStep
Output: Estimate of best epsilon - bestEpsilon
List<Double> potentialEpsilons = {};
for partition in dataPartitions do
    for p in partition do
        /* Compute the density of the areas centred in p of a radius eps multiple of
           epsStep                                                                    */
        Map<Double,Int> densities = {};
        eps=epsStep;
        density_before = 0;
        found = False;
        while (not found and eps <=distanceFromFarthestPoint(p,partition)) do
            numNeighbours = numNeighboursRadiusEps(p, eps, partition);
            density = numNeighbours/eps^d ;          /* d is the number of features */
            if (density < density_before) then
                found = True;
            end
            density_before = density
        end
        /* The potential value of epsilon for p is the one before the first density
           decrease                                                                   */
        epsilonP = eps;
        potentialEpsilons.add(epsilonP);
    end
end
/* Among the potential values of epsilon, select the one corresponding to the first
   quartile                                                                         */
bestEpsilon=FirstQuartile(potentialEpsilons);
return bestEpsilon;
```

Fig. 2. Automatic setting of the $epsilon$ parameter value.

amount of clustered data potentially decreases, while the amount of noise points increases. Since we are interested in clustering as many data points as possible, discarding only the minimum amount of objects in lower-density areas, we should consider high values of $numberOfCorePoints$. To balance the two discussed trends, we set the $MinPoints$ trade-off to the value that maximizes $MinPoints \times$ numberOfCorePoints($dataset, MinPoints, epsilon$). Figure 3 reports the pseudo-code of the algorithm that automatically sets $MinPoints$ given the value of $epsilon$. Also in this case, the procedure can be parallelized by assigning each data partition to a different Spark task.

B. Cluster and data characterization

Clusters are anonymous groups of network traffic flows; but human-readable results are much more valuable to domain experts. As such, SeLINA, as reported in block (2) of Figure 1, is designed to enrich clusters with (i) general feature-based statistics, and (ii) domain-specific knowledge, for each cluster in the resulting set, as detailed in the following. The former does not require user intervention, whereas the latter can be guided by domain experts, by a-priori selecting specific features of interest.

- *Number of flows.* It provides insights into the data distribution, by identifying clusters covering most of the dataset and others identifying small "remote" groups of traffic flows. For instance, some datasets present a predominant cluster with regular traffic and many smaller clusters identifying deviations. Other datasets may present similarly-sized clusters, (i.e., with the same number of flows) corresponding to different subnets or services.
- *Top characterizing features.* To offer the analyst the most informative features, SeLINA uses the Variance Reduction Ratio (VRR) index. Given the i-th feature x^i and an estimator for the variance $\hat{\sigma}^2$, the Variance Reduction Ratio (VRR) for the j-th cluster is defined as follows.

$$VRR_j(x^i) = \frac{\hat{\sigma}^2_D(x^i) - \hat{\sigma}^2_j(x^i)}{\hat{\sigma}^2_D(x^i)} \quad (1)$$

where $\hat{\sigma}^2_D$ is the variance over the whole dataset and $\hat{\sigma}^2_j$ is the variance over the j-th cluster. The rationale behind the variance reduction is to quantify the information gain, for a given feature, obtained by isolating some of the flows

Algorithm 2: MinPoints setting

```
Input  : Dataset partitions - dataPartitions
Input  : Epsilon - epsilon
Output: Estimate of best MinPoints - bestMinPoints
Map<Int,Int> histogramNeighbours = {};
for partition in dataPartitions do
    for p in partition do
        /* p is a core point if MinPoints is lower than or equal to the number of its
           neighbours                                                              */
        numNeighbours = numNeighboursRadiusEps(p, epsilon, partition);
        /* Update the statistics about the number of points with numNeighbours
           neighbours                                                              */
        histogramNeighbours[numNeighbours] = histogramNeighbours[numNeighbours] + 1;
    end
end
Map<Int,Int> numOfCorePoints = {};
neighboursAfterMinPoints = 0;
/* Given MinPoints, the number of core points is the number of points with more than
   MinPoints neighbours                                                            */
for MinPoints in histogramNeighbours.keys().sort().reverse() do
    numOfCorePoints[MinPoints] = histogramNeighbours[MinPoints] + neighboursAfterMinPoints;
    neighboursAfterMinPoints = numOfCorePoints[MinPoints]
end
max=0;
/* Select the MinPoints value that maximizes MinPoints × number of core points    */
for MinPoints in histogramNeighbours.keys() do
    numCorePoints = numOfCorePoints[MinPoints];
    /* d is the number of features                                                 */
    if (MinPoints > d and numCorePoints × MinPoints > max) then
        max=numCorePoints × MinPoints;
        bestMinPoints=MinPoints;
    end
end
return bestMinPoints;
```

Fig. 3. Automatic setting of the *MinPoints* parameter value.

in a cluster; it is inherited from decision trees [11], where the order of the features in the tree influences performance and results. Together with the variance itself, VRR is a strong indicator of the features that characterize a cluster the most and their relative importance.

- *Network domain statistics.* Network-oriented features of interest provided by SeLINA are the number of different source IP addresses, ports, and service types per cluster. Furthermore, the current implementation of SeLINA computes and plots the Cumulative Density Function (CDF) of selected dataset features (see Table II and Sec. V for details). For instance, per-cluster statistics of server IP addresses, server L4-ports, L7-application protocols, etc., are provided. Such features, despite being discarded during the clustering, are often crucial to allow domain experts to correctly extract meaning from the results.

C. Classification model training

All flows processed by the clustering algorithm (excluding the final iteration noise points) are labelled according to their cluster (e.g., cluster 1, 2, 3), and form a labeled dataset (i.e., a training set), which can be exploited for supervised learning. Thus, the goal of this phase, depicted in block (3) of Figure 1, is to build a classifier to efficiently label new unseen flows as they are captured.

Even if the cluster set could be directly exploited for labeling unseen data, a new ad-hoc classifier is trained separately to reach two design goals: (i) to provide a real-time high-performance classifier, and (ii) to build a human-readable model that can harness the knowledge inside the data.

To this aim, SeLINA exploits decision trees [12] to build the classification model. They are a well-known popular and mature techniques able to reach both good accuracy and easy model interpretability, with the latter being a highly-valued feature for domain experts. To provide the intuition of how a decision tree works, we describe a toy example in the

TABLE I
A TOY DATASET

Id	$RTT[ms]$	$DataByte$	Class
1	3	2M	Cl. 1
2	20	900k	Cl. 2
3	12	1.5M	Cl. 1
4	15	500k	Cl. 2
5	12	3M	Cl. 1

following.

Tree example. Table I shows a simple training set with 5 records, each characterized by two features. Two clusters/classes are present (Cl. 1 and Cl. 2). A possible decision tree is reported in Figure 4. The node labels represent a feature (e.g., the size of the flow in bytes), while each branch is labelled with a possible value, or a range of values, for the feature within the node. In our example, the split from the root node is done on a range of values of the minimum round trip time. Each path from the root node to a leaf node represents a rule characterizing a class (a cluster in our case). The path within the dotted box models the simple rule $RTT < 5ms \rightarrow cluster1$, thus this leaf can be interpreted by the analyst as a set of flows served by nearby servers, with cluster 1 partly served by those nodes. This kind of information is human-readable and provides a good characterization of how the traffic labeling is performed.

Knowledge model. The output tree provides an easy-to-read overview of the features that best split the dataset according to the labels: for each node of the tree the split criterion can be written as an if/else condition over a single feature and a splitting value, and few levels of the tree are usually sufficient to show the most significant splits for the purpose of the classification.

Split criterion. In the current work, the *Gini index* impurity-based criterion has been used to grow the tree. The Gini index is among the most popular choices and typically yields high-quality results. We exploited the Spark decision-tree implementation, which provides both the Gini and the entropy criteria. We performed some experiments, not reported here for the sake of space, to compare the accuracy of the classification models based on the Gini and the entropy indices and their results are very similar. We defer the reader to [13] for details about the Gini and the entropy indices.

IV. ONLINE MODEL UPDATE

This component analyzes new network data in real-time by applying the model built in the previous block. As depicted in Figure 1, it consists of two phases: a real-time data labeling phase (4), and an online characterization phase (5).

The classification model is exploited in the real-time data labeling phase of step (4), where each new network flow is assigned a label.

Then, at step (5) the quality index computation is executed to self-assess the model fitting over time. When the quality index falls below a given threshold, the offline model building can be automatically triggered to rebuild a model better fitting the new data, thus providing self-evolutionary features. While the online data labeling phase (4) is straightforward, as it consists of a classification model application, in the following we provide details on the quality index computation in step (5) and the self-evolution policy stemming from such quality evaluation.

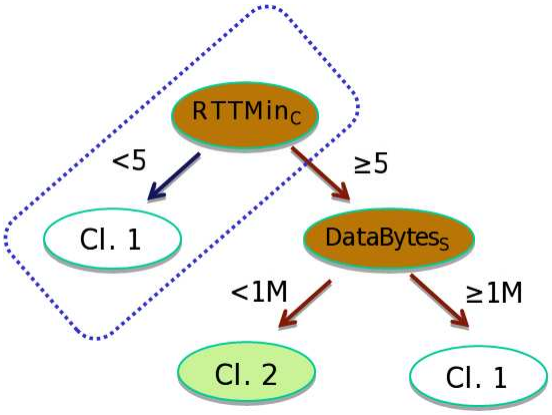


Fig. 4. A toy example of a decision tree.

A. Quality index

When no external information is provided (e.g., ground-truth class labels), the clustering results are evaluated on the shape of the clusters themselves. To this purpose, SeLINA exploits a well-known quality index, named *Silhouette* [14]. This index measures both intra-cluster cohesion and inter-cluster separation to evaluate the appropriateness of the assignment of a data object to a cluster rather than to another one.

Let $\mathbb{C} = \{C_1, \ldots, C_n\}$ be a set of clusters. The Silhouette value for a given data object r_i in a cluster $C_k \in \mathbb{C}$, given a distance measure d, is computed as

$$s(r_i) = \frac{b(r_i) - a(r_i)}{\max\{a(r_i), b(r_i)\}}, \tag{2}$$

where $a(r_i)$ is the average distance of object r_i from all other objects in cluster C_k, i.e.

$$a(r_i) = \frac{1}{|C_k|} \Sigma_{r_j \in C_k} d(r_j, r_i) \tag{3}$$

and $b(r_i)$ is the lowest average distance from all other clusters, i.e.

$$b(r_i) = \min_{C_l \in \mathbb{C}} \left(\frac{1}{|C_l|} \Sigma_{r_j \in C_l} d(r_j, r_i) \right), \forall C_l \neq C_k. \tag{4}$$

The Silhouette value for an arbitrary cluster $C_k \in \mathbb{C}$ is the average Silhouette value on all objects in C_k. It is computed as

$$s(C_k) = \frac{1}{|C_k|} \Sigma_{r_i \in C_k} s(r_i) \tag{5}$$

Lastly, the average $s(r_i)$ over all data of the entire dataset is a measure of how appropriately the data has been clustered. The distance measure d must be the same used for clustering, thus the Euclidean distance in our case.

The Silhouette coefficients take values in $[-1, 1]$. Negative and positive Silhouette values represent wrong and good object

placements, respectively. Hence, the ideal clustering algorithm splits the data in a set of clusters $\mathbb{C}$ such that all clusters in $\mathbb{C}$ have a Silhouette value equal to 1. However, Silhouette values around 0.5 are already considered very high values representing a strong clustering result [14].

B. Characterization and self-evolution policy

As the quality of the network traffic model is subject to ageing, SeLINA continuously evaluates the degradation of the model itself, with a two-fold objective: (i) highlighting substantial changes in the traffic and (ii) triggering the regeneration of the model as soon as the quality index falls below a threshold.

Since SeLINA computes the Silhouette for each new flow against the ones seen during the training phase, this quality index indicates how well the new flow fits the old clusters. A Silhouette close to 1 would indeed mean that the intra-cluster distances are negligible compared to the inter-cluster ones. The Silhouette values for the clusters ($s(C_j)$) are recomputed every N new records, where N is set by the user. The Silhouette index for the clusters significantly changes as soon as new kinds of data (not seen during the training) are added to the input (see Section VII-B for an example). Unseen values should indeed get a Silhouette close to 0, while mispredictions, i.e., assignments to the wrong cluster, would have a negative value.

Besides the Silhouette indicator, SeLINA also tracks the number (percentage) of new flows assigned to each cluster over time. This helps in detecting changes in the traffic characterization due to (i) degradation in the clustering quality and (ii) shift in the distribution of the traffic flows among different clusters, as discussed in the experiments.

The final goals of the real time evaluation are to keep track of the state of the network, to identify changes and react. The reaction strongly depends on the use case and on the type of change. When SeLINA is trained on a standard behaviour, e.g., a usual working day without interruptions of service or congestion, a change is a strong hint of an anomaly, a strong congestion or an attack. The identification can be performed by looking at the current clusters and their cardinality in recent time frames. If the Silhouette value is unchanged, the current clusters do still model well the traffic, and the anomaly occurs only in the distribution of the flows among the clusters. If the Silhouette value of one or more clusters decreases, instead, the change is way more significant: the current model cannot describe the traffic anymore. The inspection in this case needs a new clustering, which can also be automatically triggered by the system. The new clustering can be executed on the whole historical dataset or on the most recent flows only. The latter option generates a more specific up-to-date model, that could be less general due to fewer training data.

V. Experiments and datasets

We experimentally evaluated SeLINA on two real network traffic datasets, associated with two different use cases. Our goal is to show how SeLINA (i) effectively characterizes network traffic, and (ii) supports the analyst in understanding changes of the traffic mix. We focused on two real-world use cases. The first one consists of a dataset of YouTube flows in which we know the CDN had changed over time, causing possible issues to both end-users and ISPs [3]. The second case deals with the understanding of P2P traffic, for which, instead, little knowledge is available. In both cases, SeLINA autonomously extracts information from the automatic analysis of the traffic summaries, and presents results to domain experts in an interpretable format.

We collected network traffic data through a passive probe located on the access link (vantage point) connecting an ISP Point of Presence (PoP) to the Internet. The passive probe sniffs all packets flowing on the link. The probe runs Tstat [15], a passive monitoring tool that extracts flow level logs. Tstat rebuilds each TCP (and UDP) stream by matching incoming and outgoing segments (and messages). A flow-level analysis is performed, and for each flow a set of metrics is logged [7]. Tstat offers advanced classification mechanisms that we leveraged to split traffic according to the application that generated it.

In this work, we focus on two datasets, collected during two different time periods. The first one consists of flows carrying YouTube videos. The second one collects all TCP flows excluding web traffic, i.e., it consists of mostly P2P traffic. We refer to each dataset as "YouTube" and "P2P" in the following.

The YouTube dataset consists of TCP flows collected during May 2013 by a probe placed on a PoP of a nation-wide ISP in Italy where the traffic aggregate from more than 10,000 customers is monitored. We use data from May 1st, 2013 to let SeLINA build the offline model. Datasets from May 2nd to May 31st are used instead to run the online model update phase and highlight traffic changes possibly suggesting the automatic model rebuilding. For this dataset, we know that during the second part of May 2013 the YouTube CDN had relevant changes affecting end-user quality of experience [3], [1]. Hence, we consider this as ground-truth information that allows us to verify if SeLINA correctly identifies interesting events.

The P2P dataset refers to April 17, 2012. From it we extract all TCP flows whose application protocol is neither HTTP nor HTTPS, i.e., where the majority of the traffic is due to P2P applications [16]. Traffic comes again from a backbone link of a nation-wide ISP in Italy.

Among the measurements exposed by Tstat, we consider the metrics reported in Table II. We selected them since they are correlated to both system configuration and possible performance issues. For instance, the measure of the Round Trip Time (RTT) is related to both the distance from the server, and possible congestion on the path. Similarly, both reordering and duplicate probabilities increase during periods of congestion. Finally, duration and amount of carried data are possibly linked to the type of service the flow carries, e.g., short-lived signaling flows carrying little data rather than long lived data flows carrying a large amount of data. Since SeLINA model building is based on unsupervised clustering, we expect the system to automatically leverage information offered by these features to identify proper classes of flows.

TABLE II
FEATURES USED BY SELINA AS INPUT.

Metric	Description	Intuition
$L7-Data$	Amount of application payload transferred	Identifies possible different type of flows, e.g., data vs signaling
$Duration$	Time since the first SYN to the last segment	Related to performance issues, and type of flow, e.g., bulk transfer or persistent connections
$RTTMin$	Per-flow minimum RTT	Estimate of the "distance" between client and server, and of possible congestion
P_{reord}	Per-flow reordering probability	Identifies possible packet losses occurred before the probe
P_{dup}	Per-flow duplicate probability	Identifies possible packet losses occurred after the probe

The metrics are computed by observing the TCP headers, and correlating them with information in the corresponding TCP ACKs. For instance, P_{reord} and P_{dup} are computed by keeping track of TCP sequence number evolution over time, while $RTTMin$ is the minimum delay observed between a data segment and the corresponding acknowledgement. Since TCP offers a bidirectional service, we consider measurements for each half-flow, i.e., segments from the client to the server, and vice versa. We denote them in the following by adding a subscript C or S for client or server side, respectively. For instance $RTTMin_S$ is the minimum delay observed at the probe between segments sent by the server and ACKs sent by the client, i.e., it is the delay between the probe and the customer client – the access network delay. Conversely, $RTTMin_C$ measures the time since the probe observes the client segment and the server ACK, i.e., it is the minimum RTT between the probe and the server – the backbone network delay.

Additional features and measures are included in the final results flows aggregated in the same cluster. Clusters are annotated by SeLINA before being presented to the domain experts. The additional features are not considered during the model building phase. For instance, once the cluster is built, the system computes Cumulative Density Functions (CDF), average, percentiles, etc. of the per-metric distribution of information extracted directly from features.

The datasets have been stored in a cluster at our University running Cloudera Distribution of Apache Hadoop (CDH5.3.1). All experiments have been performed on our cluster, which has 30 worker nodes, and runs Spark 1.2.0, HDFS 2.5.0, and Yarn 2.5.0. The cluster has a total of 2.5TB of RAM, 324 cores, and 773TB of secondary memory. The current implementation of SeLINA is a project developed in Scala exploiting the Apache Spark framework.

VI. YOUTUBE USE CASE

In this section we discuss the network traffic characterization of the YouTube dataset first, as a result of the offline SeLINA component, and then we present an evaluation of the online part. The default values of $EpsStep$=0.001 and 3 clustering levels led to meaningful results for this experiment. Increasing the number of levels brings no improvement. After the third iteration, new clusters become very small and have very low Silhouette values, a clear sign that the system is artificially aggregating data that are actually very fragmented.

A. Offline cluster and model characterization

Clustering results provide meaningful insights into network traffic when enriched by means of relevant statistics and features. As such, we present traffic analyses provided by both the cluster statistics and the classification model.

1) Cluster statistics: Table III reports the clusters obtained by running SMDBScan on the YouTube dataset of May 1st, 2013. For each cluster, SeLINA returns the top-3 features according to VRR, i.e., it presents to the network analyst those features that best represent the data inside the cluster itself. For instance, consider the cluster number 1. It is the biggest one, collecting approximately 60,000 (36%) flows. It is primarily characterized by a rather low P_{dup} value (0.65%±0.71%), and clients requesting 4kB of data on average ($L7-Data_C$=3992±2422.4 bytes), a rather sizable HTTP request size. $RTTMin_S$ is 33.7±16.1 ms, which suggests quite standard and not congested DSL lines. The cluster thus collects the most common flows. This is the only cluster identified during the first iteration of the multi-level clustering. During iteration 2 and 3, more clusters emerge (one in step 2, and two in step 3), each with several thousands of flows. This confirms the ability of SMDBScan to identify large clusters, despite different densities, thanks to the multi-level approach. At the end of the whole clustering process, the noise cluster aggregates all remaining points. There are 40,000 of them (23%), which are very sparse, as proven by the high variance in their characterizing features.

Clusters 2 and 3 represent a sizable part of the traffic, with 16% and 22% of the flows, respectively. Interestingly, those are characterized by two very different $RTTMin_C$ values. Recall that $RTTMin_C$ represents the distance of the YouTube CDN server to the probe. Servers in Cluster 2 are 25.3±1.4 ms far, while servers in Cluster 3 are much closer (5.4±4 ms). P_{dup} is significantly different too, with Cluster 2 P_{dup} being one order of magnitude smaller than cluster 3. This probably reflects higher congestion in the path from the probe to the client in cluster 2.

Cluster 4 collects fewer points (5,500, 4%). P_{dup} and $RTTMin_C$ are similar to Cluster 2, but here duration (51±32 s) is very large. This possibly hints for TCP flows of long lived video sessions. We will get back to this when observing the video resolution distribution in the following.

Besides the top-3 features selected for each cluster, SeLINA offers to the analyst a further characterization of the network traffic by presenting CDFs of features and additional measurements collected by the probe. In this use case, we consider the distribution of the RTT, of the throughput, and of the type of video format and resolution.[2]

Fig. 5 and Fig. 6 report the CDF of the $RTTMin_C$ and the average download throughput for each cluster. Cluster 2 and

[2]Tstat has a DPI engine specialized in extracting metadata from YouTube flows.

TABLE III
YOUTUBE DATASET. CLUSTER CHARACTERIZATION.

Lev.	Cl. id	Num. flows	Top-3 representative features ranked by highest variance reduction ratio		
			Feature	Avg. value	Std. dev
1	1	59846	P_{dup}	0.65%	7.12E-03
			$L7-Data_C$	3992.1	2422.4
			$RTTMin_S$	33.7	16.1
2	2	27158	$RTTMin_C$	25.3	1.4
			P_{dup}	0.55%	6.42E-03
			$L7-Data_C$	5357.8	2916.9
3	3	37964	P_{dup}	2.97%	2.31E-02
			$RTTMin_C$	5.4	4.0
			$RTTMin_S$	52.6	42.2
	4	5569	$RTTMin_C$	25.5	1.2
			P_{dup}	4.11%	1.28E-02
			$Duration$	51464.0	31969.4
noise		40318	P_{reord}	0.000002%	3.12E-06
			$L7-Data_S$	14465449.3	30919388.4
			$RTTMin_S$	78.7	160.6

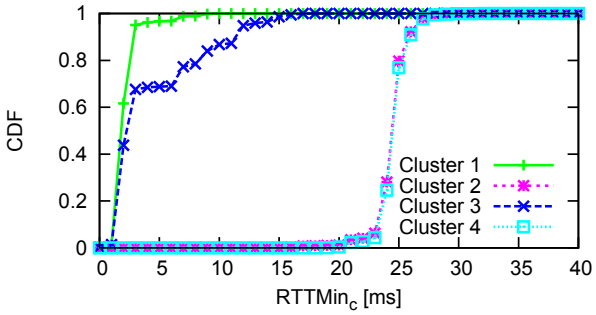


Fig. 5. YouTube dataset. $RTTMin_C$ distribution for each cluster.

4 show similar distributions of the $RTTMin_C$, as previously discussed, which are significantly different from the clusters 1 and 3, whose flows, instead, are characterized by generally low values of $RTTMin_C$ (i.e., they represent requests served by nearby CDN servers). On the contrary, the flows of clusters 2 and 4 are associated with video requests that are served by relatively far CDN servers. Figure 6 shows that cluster 4 is also characterized by worse performance in terms of throughput, and it probably represents flows with possible performance issues.

This reflects the typical scenario of the YouTube CDN [3], and proves the ability of SeLINA to provide insights on the traffic mix. The analyst is offered few and consistent clusters, instead of thousands of single measurements.

To investigate further the characteristics of each cluster, Table IV details the percentage of flows per cluster for each video resolution format. For each cluster, the 3 most frequent formats are reported. Each format is characterized by the quality of the video, mainly in terms of resolution (e.g., 240p, 720p), and it is identified by an integer value.[3] Some formats are shared by all clusters (i.e., format id 34 and 134), whereas others are peculiar for specific clusters, such as format 25 for

[3]Video Resolution information at https://en.wikipedia.org/wiki/YouTube.

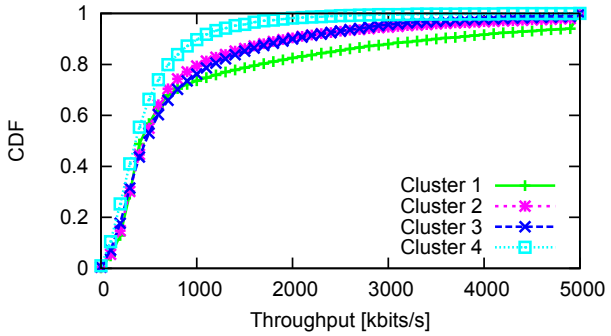


Fig. 6. YouTube dataset. Throughput distribution for each cluster.

TABLE IV
YOUTUBE DATASET. CLUSTERS' CHARACTERIZATION BASED ON VIDEO FORMAT

Lev.	Cl. id	Top-3 format ranked by number of flows	
		Format id (resolution)	Num. of flows (%)
1	1	25	63.71%
		34 (360p)	10.29%
		134 (360p [DASH])	7.36%
2	2	34 (360p)	60.67%
		134 (360p [DASH])	9.02%
		35 (480p)	8.35%
3	3	34 (360p)	55.12%
		35 (480p)	9.34%
		134 (360p [DASH])	9.22%
	4	34 (360p)	50.26%
		140 (AAC 128)	10.56%
		134 (360p [DASH])	9.43%

TABLE V
QUALITY OF THE CLASSIFICATION ALGORITHM. 3-FOLD CROSS-VALIDATION

Cluster id	Precision	Recall
1	96.28%	80.17%
2	97.73%	93.02%
3	97.91%	99.25%
4	88.93%	98.22%

cluster 1, and format 140 for cluster 4. It is interesting to notice that neither the format id, nor any other video information was exploited during the clustering phase, nevertheless the clusters are correlated to a set of video formats. For cluster 4, for instance, the longer *Duration* is due to higher presence of high quality audio stream server in format 140 (AAC 128kbps) that is not found in other clusters.

2) Classification rules: The decision-tree described in Section III-C and trained with a maximal depth of 4 levels has been evaluated with a 3-fold cross-validation scheme on the training data. The average accuracy over the three cross-validation runs is 93%, and results for precision and recall for each class are shown in Table V. All clusters are extremely well represented by the model for both precision and recall (93% to 99%), apart from a lower value in Cluster 1 recall (80%).

Being the model so accurate, rules that form the decision tree can be used to understand how the different clusters split the network traffic. Each path from the root to one leaf of the

decision tree is translated into a rule for the class of that leaf. Each rule characterizes the data of its class (i.e., a cluster in our case). Rule-based modeling provides further insights into correlations among features. Analying the rules of such classifier, we observe the following:

- $\{(RTTMin_C > 15.7ms)\ and\ (P_{dup} > 2.5\%)\} \rightarrow Cluster4$. This is the only rule associated with cluster 4. It states that all flows of cluster 4 are simultaneously characterized by a high $RTTMin_C$ and a high P_{dup}.
- $\{(RTTMin_C > 21.5ms)\ AND\ (P_{dup} \leq 2.5\%)\} \rightarrow Cluster2$. This rule provides a characterization of cluster 2, where flows have an even higher $RTTMin_C$ but a lower P_{dup} with respect to cluster 4.

Insights provided by such rules are relevant since we would not have been able to distinguish the differences between clusters 2 and 4 by considering only the CDF of $RTTMin_C$ reported in Figure 5. The rules, which simultaneously consider more than one measure, allow supporting domain experts to more easily characterize the content of the clusters and also perform comparisons among them by considering at the same time many facets.

B. Online data characterization and model update

The decision tree model is exploited to assign new flows, in real time, to the most appropriate class (which is one of the clusters). Every N assignments, SeLINA evaluates the quality of the current cluster set, by means of the Silhouette quality index and the distribution in number of flows assigned to each cluster. These two indicators provide the self-evolution feature to SeLINA, which is able to trigger a model rebuilding phase. The analysis of the Silhouette index indicates whether the classifier model does not fit the current data anymore, and the distribution of the flows among the clusters indicates whether the traffic patterns are changing. This information is also valuable for the analyst since it reflects changes in the traffic mix.

The upper part of Figure 7 reports the value of the Silhouette index for three different days (May 2nd, May 3rd, and May 29th from left to right). The lower part reports the percentage of flows assigned to each cluster over time. The values are computed every $N = 10,000$ flows, which corresponds to 2-3 hours at night, and approximately 1 hour during peak traffic hours. Recall that the model had been trained on the May 1st dataset. Traffic from following days is assigned to clusters based on the classifier, but without re-running the clustering itself.

As discussed by domain experts in [3], network traffic in the first part of May is very similar to May 1st. On the contrary, in the second part of May, a change in the YouTube CDN occurred. As such, we would expect the Silhouette to reflect this situation, especially during peak time when the traffic is more significant. This is indeed the case. The Silhouette value is rather stable for all clusters during the first days of May, of whom we reported here May 2nd and May 3rd, meaning that there are no important changes in the traffic with respect to the May 1st model. It still fits the new data. Only cluster 2 and 4 show temporary and limited drops in Silhouette values from 10am to 12am, but at that time, they only account for very low percentages of the traffic (less than 10% each).

The Silhouette values of clusters 2 and 4 during May 29th, when the significant change in the YouTube CDN already occurred, drop suddenly from 12pm to 8pm of May 29th. At that time, a sizable amount of traffic is assigned to these two clusters ($\approx$ 20% each), but the clusters 2 and 4 do not fit the data anymore, so that new flows present un-modeled characteristics, and they fall into the low-Silhouette clusters. Interestingly, cluster 1 still has a high Silhouette value (and counts for 30-40% of the traffic), reflecting that not all traffic is affected by the change.

A detailed analysis of May 29th highlights a significant increase of the $RTTMin_C$ values for cluster 2 and cluster 4 flows. While in May 1st and May 2nd, almost all flows have an $RTTMin_C$ lower than 25ms, in May 29th there are many flows with $RTTMin_C$ from 80ms to 100ms. The increase of the $RTTMin_C$ values is associated with changes in the YouTube's CDN previously identified in [3]. In the model built by SeLINA on May 1st data, there are no clusters representing this traffic pattern. Thus, the sudden drop of the Silhouette values automatically highlights the changes and can be used to raise an alarm.

To better highlight the difference in SeLINA results, we ran a set of experiments considering the first and last five days of May. We aim at identifying groups of similar traffic days, by means of the Silhouette pattern. Fig. 8 shows the correlation matrix of the per-day Silhouette indexes, i.e., for each pair of days, we measure how similar the Silhouette trends are. We use the Pearson correlation coefficient [13] among the Silhouette values during two days: when close to 1, the Pearson correlation depicts a strong positive linear correlation; when the coefficient is close to -1, it highlights a negative correlation, and when it is close to 0, negligible correlation is found. Left plot of Fig. 8 clearly highlights that Cluster 1 Silhouette is always very similar among those days, and stable over time. Cluster 1 consistently represents the part of traffic not affected by the CDN change, and the model always fits the new data. Right plot of Fig. 8, instead, clearly shows that Cluster 4 exhibits two patterns over time: during the beginning of May, it is consistent with the model. But during the last part of May (after the CDN change), its Silhouette daily pattern becomes very different from before. These results prove the strong link between the change in the Silhouette trends and the change in the traffic patterns, and the validity of using the drop of the Silhouette as a trigger for the generation of a new model of the network.

Focusing on the percentage of new flows assigned to each cluster (bottom plots in Fig. 7), we can detect changes related to the CDN allocation policy. For all days, the distribution of the flows changes from the late morning till evening. In particular, Cluster 1 traffic, which is characterized by low $RTTMin_c$ values, decreases from 60-70% (at night) to 30-40% in the 10am-9pm period. On the contrary, the number of flows assigned to Clusters 2 and 4 increases from less than 5% to 20-30% each. Clusters 2 and 4 are characterized by higher $RTTMin_c$ values, i.e., traffic is now being served by far-away

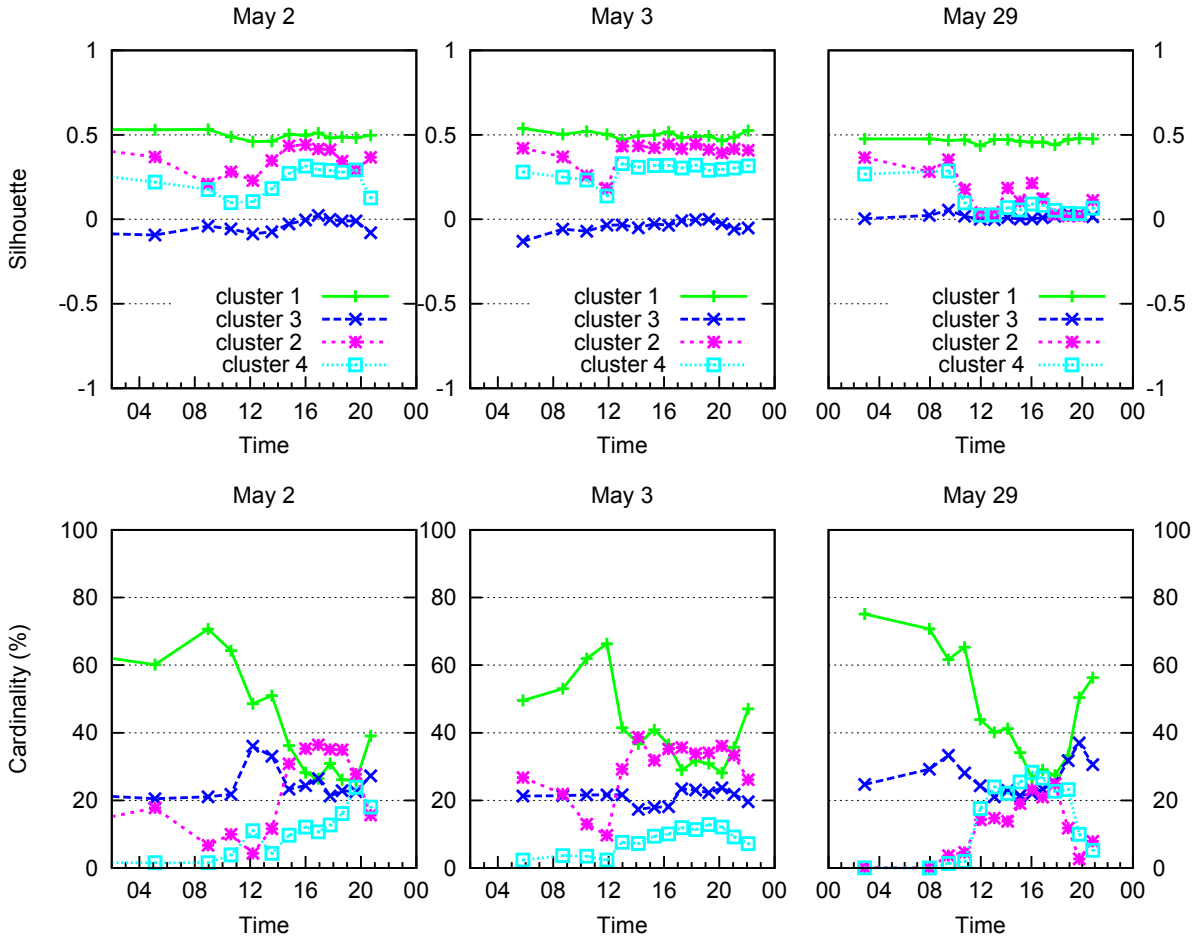


Fig. 7. YouTube dataset. Real-time data labeling: Silhouette and percentage of new flows assigned to each cluster.

servers. The difference between the two days is that in May 2nd and May 3rd the Silhouette values are high and stable, hence the changes in the distribution of the flows among the clusters are meaningful. On the contrary, on May 29th the drop in Silhouette values means that clusters cannot be trusted, and thus a model rebuilding is required.

We let then SeLINA rebuild the whole model (clustering and classifier) on the flows of May 29th from 10am to 9pm. This leads to a new characterization of the network traffic, not shown here due to lack of space. 10 clusters (instead of 4) are identified, with very different characterizing features, both in terms of number of flows and statistical distribution of values. The new model includes some clusters with very high $RTTMin_C$ values, which means that the presence of new CDN servers that were not properly represented by the previous clusters are now covered. For example, one of the new clusters, which contains about 12,000 flows, is characterized by an average $RTTMin_C$ value of 99ms±11ms, a significantly higher value than those of the former clusters (see Fig. 5). These results are consistent with those in [3], and confirm the ability of SeLINA to automatically identify changes in traffic pattern. Moreover, SeLINA extracts clusters which fit the new data and provide insightful analyses of the network traffic evolution.

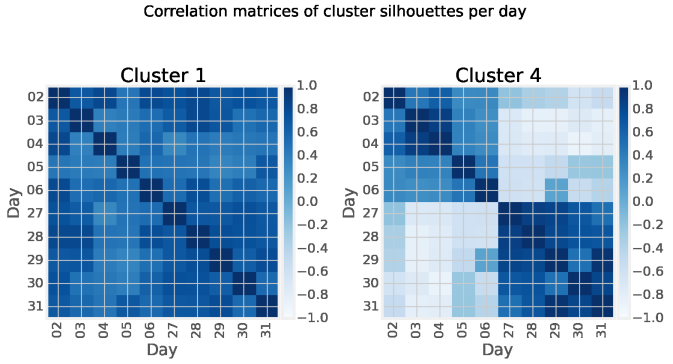


Fig. 8. YouTube dataset. Per-day correlation matrices of silhouettes for cluster 1 and 4.

VII. P2P USE CASE

In this section we show how SeLINA can help characterize the flows of the P2P dataset. We recall that this dataset contains all the TCP flows captured by Tstat, except the HTTP and HTTPS ones. Also in this case, we executed the offline self-learning phase by using the default values of $EpsStep$=0.001 and the first 3 levels of clustering. Differently from the YouTube use case, here we have no ground truth at

TABLE VI
P2P DATASET. CLUSTER CHARACTERIZATION. CLUSTERS OBTAINED BY USING SMDBSCAN AND SETTING $EpsStep$=0.001

Lev.	Cl. id	Num. flows	Top-3 representative features Ranking by highest variance reduction ratio		
			Feature	Avg. value	Std. dev
1	1	98186	$RTTMin_S$	33.5	38.6
			$RTTMin_C$	35.0	45.7
			$Duration$	33154.1	43104.8
2	2	15090	P_{dup}	3.30E-06	2.04E-04
			$RTTMin_S$	235.5	74.8
			$RTTMin_C$	9.4	22.2
	3	12152	P_{dup}	2.37E-05	5.44E-04
			$RTTMin_S$	14.6	22.9
			$RTTMin_C$	295.3	72.9
	4	4530	P_{dup}	4.44E-01	1.34E-03
			$RTTMin_S$	11.2	16.2
			$RTTMin_C$	18.5	12.9
3	5	3302	P_{dup}	1.31E-05	5.07E-04
			$RTTMin_S$	542.0	103.6
			$RTTMin_C$	5.2	13.6
	6	2524	P_{dup}	2.80E-01	2.49E-02
			$RTTMin_S$	6.9	15.0
			$RTTMin_C$	32.5	37.9
	7	1993	P_{dup}	1.05E-05	3.65E-04
			$RTTMin_S$	16.9	28.9
			$RTTMin_C$	608.1	101.5
	8	1892	P_{dup}	1.60E-01	1.88E-02
			$RTTMin_S$	14.7	29.6
			$RTTMin_C$	35.6	42.1
noise		13647	$Duration$	560334.5	4671692.5
			$L7-Data_S$	3585728.2	27938244.0
			P_{reord}	5.05E-03	2.90E-02

our disposal, and thus SeLINA is used as a data exploration tool.

A. Offline cluster and model characterization

Table VI reports the main characteristics of the extracted clusters and their top-3 characterizing features. We immediately notice a cluster with approximately 64% of the flows (Cluster 1, 98186 flows) that is significantly larger than any other cluster. Cluster 1 represents "standard" flows which are characterized by a similar average value of $RTTMIN_C$ and $RTTMIN_S$ (i.e. the communication time is similar in both directions of the flow). This balancing between $RTTMIN_C$ and $RTTMIN_S$ values is normal when no congestions are present. Indeed, P2P traffic is exchanged between residential clients, therefore, we can expect the distance between the probe and the peers to be somewhat equal, and so the RTT.

Other clusters provide interesting knowledge to domain experts as well. For instance, Cluster 2 has an average $RTTMin_S$ (235.5ms) that is two orders of magnitude higher than $RTTMin_C$ (9.4ms). These values highlight a significant asymmetry between the server side and the client side. The $RTTMin_S$ is very high (the average $RTTMin_S$ value of the 'standard" flows in Cluster 1 is 33.5ms). This situation describes a possible congestion in one direction of the communication flow. We recall that we are analyzing P2P flows among ISP customers where many users are connected through an ADSL connection, with uplink capacity limited to 1Mbps. When a remote peer downloads a large amount of data from a local peer, the uplink of the latter may saturate, causing congestion (i.e., the average $RTTMin_S$ increases). On the contrary, the small $RTTMin_C$ suggests that the remote peer is connected via high speed FTTH technology (where access delay is much smaller being the upling capacity >10Mbps). A similar situation is valid also for the flows of Cluster 7. However, in Cluster 7 the $RTTMin_C$ is very high (608ms) and the $RTTMin_S$ is low (16.9ms). This second case reflects a simmetric scenario: high-speed local client downloading a lot of data from ADSL remote peers, whose uplink results congested.

B. Online data characterization and model update

For the P2P dataset we applied the evolving part of the framework to analyze how new flows are assigned to the clusters and identify possible changes in the type of traffic. Results present no significant changes in the Silhouette. This trend, that is confirmed by further statistics computed on the flows, highlights that there are no anomalies or changes in the traffic for the P2P dataset. The clusters identified by the clustering phase are still representative of the network flows, also of the future traffic. Since the day we analyzed is a "normal" one, SeLINA correctly identifies no changes and hence the re-execution of the clustering phase is not triggered.

VIII. RELATED WORK

A significant effort has been devoted to the application of data mining techniques and statistical methods to network traffic analysis. The application domains include studying correlations among network data (e.g., association rule extraction for network traffic characterization [17], [18]; for router misconfiguration detection [19]; interesting correlations from web-based e-business system [20]), extracting information for prediction (e.g., multilevel traffic classification [21], Naive Bayes classification [22], throughput prediction [23], analytics and statistical models for LTE Network Performance [24], one-class SVM [25] for intrusion detection), grouping network data with similar properties (e.g., clustering algorithms for intrusion detection [26], [27], [28], [4], [5], for deriving node topological information [29], for automatically identifying classes of traffic [30], [31], [32], [33], for unveiling YouTube CDN changes [3]), and context specific applications (e.g., multi-level association rules in spatial databases [34]).

However, in most cases no approach offloads the user from arbitrary parameter choices, and can be easily adapted to domain-specific requirements and semantics as the methodology proposed in this paper. Differently from analytics approaches tailored to a specific network application [3], [26], [27], [29], [28], [4], [5], SeLINA is a general purpose methodology that can be easily exploited to analyze different and transversal network data (e.g., network traffic headers, network flow characteristics, statistical measurements of traffic flows). In the experimental section we considered Youlighter [3]. Youlighter is a system that detects very specific macro-changes in the YouTube traffic pattern involving the CDN spatial distribution. It is not distributed nor scalable. SeLINA, instead, introduces a general-purpose, distributed,

and fully autonomous engine exploiting a completely different methodology and addressing a more general research issue in the network traffic analysis than the Youlighter system.

The performance of most state-of-the-art general purpose approaches [17], [18], [30], [31], [32], [33] depends on the choice of different parameters, and the optimal trade-off between execution time and accuracy must be handpicked for a given application. On the contrary, focusing itself on self-learning capabilities of state-of-the-art scalable approaches, SeLINA is able to build a model of the data with minimal user intervention by offloading the user from the non trivial task of configuring the miner system and highlighting possibly meaningful interpretations to domain experts.

Some research effort has been devoted to automatic setting of data mining algorithm parameters (e.g., clustering algorithms [35], [36], itemset mining [37]). Authors in [35], [36] proposed a hierarchical strategy to aggregate lower density regions discovered through DBSCAN. Different from [35], [36], SMDBScan automatically sets DBScan parameters at each iteration level when DBScan is exploited in a multiple-level fashion. Furthermore, the SeLINA clustering results include clusters with a diverse degree of density, because each subset of clusters with a similar density is discovered at a given iteration level. The method in [35], [36] instead gets a flat partition composed of clusters extracted from local cuts through the cluster tree.

An intensive research activity has been devoted to designing innovative algorithms and methodologies to support large scale analytics based on MapReduce, such as [38], [39], [40]. A step further has been proposed in [41]. Apache Spark with its Resilient Distributed Datasets and its smart APIs, outperforms Hadoop MapReduce in terms of performance and overcomes its limitations, with particular focus on iterative in-memory computation, which is a common characteristic of many data mining algorithms. Its machine learning library MLlib [42] provides a broad range of analytics algorithms. SeLINA exploits the computational advantages of distributed computing frameworks, as the current implementation runs on Spark. Applications of this techniques to network traffic analysis becomes natural, given the volume of traffic [6], [43], [44], [45]. These works adopt Hadoop or Spark, and apply either standard machine learning algorithms, or design specific solutions to their problem.

The idea of defining a generic framework and of tightly integrating self-learning capability in a scalable data mining engine tailored to traffic data was first introduced by ourselves in [46]. However, SeLINA significantly enhances the methodology proposed in [46]. The SeLINA data mining engine (named SaFe-Nec in [46]) provides an innovative and more accurate explorative approach coupled with self-configuring strategies (i.e., the SMDBScan algorithm). Thus, SeLINA allows exploiting cluster analysis on real datasets in a fully autonomous fashion. The SMDBScan algorithm is characterized by configuration parameters whose setting is rather difficult. In [46] the less effective, but easier to configure, K-means algorithm was used. SeLINA also includes ad-hoc strategies to automatically tune the clustering parameter values, which is a typically difficult task also for domain experts. The exploitation of a multilevel DBScan-like algorithm jointly with self-configuring strategies allowed for better clustering results than the ones produced by the K-means based approach proposed in [46]. Moreover, SeLINA integrates innovative self-assessment features and a new set of network domain statistics that are often vital to let the domain expert interpret the results. Finally, with respect to [46], in this paper we added a new interesting case study on YouTube traffic analysis and a thorough analysis of the results from a networking point of view.

IX. Conclusion

This paper presents a self-learning data analytics system that effectively mines network traffic data. The proposed methodology is based on a two-phase approach that

1) builds a self-evolving human-readable traffic model by autonomously splitting traffic data into homogeneous groups;
2) classifies new data in real-time and identifies the presence of changes in the traffic mix.

The SeLINA methodology features a distributed implementation in Apache Spark. It is a general purpose approach, which can be easily exploited to analyze network traffic data in different conditions. The approach has been tested in two real-world use cases. The performed experiments highlighted its ability to autonomously identify evolutions in the network and support the analyst by selecting characterizing features.

Possible extensions of the current work are (i) the inclusion of further cluster characterization measures, (ii) the evaluation of pre-processing feature selection techniques, and (iii) the design and integration of different analysis techniques, more appropriate for outlier detection.

Acknowledgment

The authors would like to thank Luigi Celona and Marco Gaido for implementing portions of the SeLINA system. The research leading to these results has received funding from the European Union under the FP7 Grant Agreement n. 619633 (“ONTIC” Project).

References

[1] P. Casas, A. D’Alconzo, P. Fiadino, A. Bär, A. Finamore, and T. Zseby, “When youtube does not work - analysis of qoe-relevant degradation in google CDN traffic,” *IEEE Transactions on Network and Service Management*, vol. 11, no. 4, pp. 441–457, 2014.

[2] A. Bär, A. Finamore, P. Casas, L. Golab, and M. Mellia, “Large-scale network traffic monitoring with dbstream, a system for rolling big data analysis,” in *2014 IEEE International Conference on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014*, 2014, pp. 165–170.

[3] D. Giordano, S. Traverso, L. Grimaudo, M. Mellia, E. Baralis, A. Tongaonkar, and S. Saha, “Youlighter: A cognitive approach to unveil youtube cdn and changes,” *IEEE Transactions on Cognitive Communications and Networking*, vol. 1, no. 2, pp. 161–174, June 2015.

[4] P. Casas, J. Mazel, and P. Owezarski, “Unsupervised network intrusion detection systems: Detecting the unknown without knowledge,” *Computer Communications*, vol. 35, no. 7, pp. 772–783, 2012.

[5] J. Dromard, G. Roudiere, and P. Owezarski, “Unsupervised network anomaly detection in real-time on big data,” in *New Trends in Databases and Information Systems - ADBIS 2015 Short Papers and Workshops, BigDap, DCSA, GID, MEBIS, OAIS, SW4CH, WISARD, Poitiers, France, September 8-11, 2015. Proceedings*, 2015, pp. 197–206.

[6] Y. Lee and Y. Lee, "Toward scalable internet traffic measurement and analysis with hadoop," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 1, pp. 5–13, 2013.

[7] M. Mellia, M. Meo, L. Muscariello, and D. Rossi, "Passive analysis of tcp anomalies," *Computer Networks*, vol. 52, no. 14, pp. 2663–2676, 2008.

[8] J. Han, *Data Mining: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[9] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, 1996, pp. 226–231.

[10] D. Antonelli, E. Baralis, G. Bruno, T. Cerquitelli, S. Chiusano, and N. A. Mahoto, "Analysis of diabetic patients through their examination history," *Expert Syst. Appl.*, vol. 40, no. 11, pp. 4672–4678, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2013.02.006

[11] L. Rokach and O. Maimon, *Data Mining with Decision Trees: Theroy and Applications*. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 2008.

[12] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[13] Pang-Ning T. and Steinbach M. and Kumar V., *Introduction to Data Mining*. Addison-Wesley, 2006.

[14] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53 – 65, 1987. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0377042787901257

[15] A. Finamore, M. Mellia, M. Meo, M. Munafò, and D. Rossi, "Experiences of internet traffic monitoring with Tstat," *IEEE Network*, vol. 25, no. 3, pp. 8–14, 2011.

[16] J. L. Garcia-Dorado, A. Finamore, M. Mellia, M. Meo, and M. Munafo, "Characterization of isp traffic: Trends, user habits, and access technology impact," *IEEE Transactions on Network and Service Management*, vol. 9, no. 2, pp. 142–155, June 2012.

[17] D. Apiletti, E. Baralis, T. Cerquitelli, and V. D'Elia, "Characterizing network traffic by means of the netmine framework," *Computer Networks*, vol. 53, no. 6, pp. 774–789, 2009.

[18] M. Hossain, S. Bridges, and R. Vaughn Jr, "Adaptive intrusion detection with data mining," *IEEE Internation Conference on Systems, Man and Cybernetics*, vol. 4, 2003.

[19] F. Le, S. Lee, T. Wong, H. S. Kim, and D. Newcomb, "Minerals: using data mining to detect router misconfigurations," in *MineNet '06*. New York, NY, USA: ACM Press, 2006, pp. 293–298.

[20] M. K. Agarwal, M. Gupta, G. Kar, A. Neogi, and A. Sailer, "Mining activity data for dynamic dependency discovery in e-business systems," *IEEE Transactions on Network and Service Management*, vol. 1, no. 2, pp. 49–58, 2004.

[21] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "Blinc: multilevel traffic classification in the dark." in *SIGCOMM*, 2005, pp. 229–240.

[22] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *SIGMETRICS '05*. New York, NY, USA: ACM Press, 2005, pp. 50–60.

[23] J. E. B. Maia *et al.*, "Network traffic prediction using pca and k-means," in *Network Operations and Management Symposium (NOMS), 2010 IEEE*. IEEE, 2010, pp. 938–941.

[24] Y. Ouyang, M. H. Fallah, S. Hu, Y. R. Yong, Y. Hu, Z. Lai, M. Guan, and W. Lu, "A novel methodology of data analytics and modeling to evaluate LTE network performance," in *2014 Wireless Telecommunications Symposium, WTS 2014, Washington, DC, USA, April 9-11, 2014*, 2014, pp. 1–10.

[25] M. Amer, M. Goldstein, and S. Abdennadher, "Enhancing one-class support vector machines for unsupervised anomaly detection," in *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*, ser. ODD '13, 2013, pp. 8–15.

[26] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering," *Proceedings of ACM CSS Workshop on Data Mining Applied to Security, PA,, November*, 2001.

[27] Q. Wang and V. Megalooikonomu, "A clustering algorithm for intrusion detection," *Proc. SPIE*, vol. 5812, pp. 31–38, 2005.

[28] P. Owezarski, "Unsupervised classification and characterization of honeypot attacks," in *10th International Conference on Network and Service Management, CNSM 2014 and Workshop, Rio de Janeiro, Brazil, November 17-21, 2014*, 2014, pp. 10–18.

[29] E. Baralis, A. Bianco, T. Cerquitelli, L. Chiaraviglio, and M. Mellia, "Netcluster: A clustering-based framework to analyze internet passive measurements data," *Computer Networks*, vol. 57, no. 17, pp. 3300–3315, 2013.

[30] L. Grimaudo, M. Mellia, E. Baralis, and R. Keralapura, "Select: Self-learning classifier for internet traffic," *IEEE Transactions on Network and Service Management*, vol. 11, no. 2, pp. 144–157, 2014.

[31] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms," in *MineNet '06*. New York, NY, USA: ACM Press, 2006, pp. 281–286.

[32] J. Y. Chung, B. Park, Y. J. Won, J. Strassner, and J. W. Hong, "An effective similarity metric for application traffic classification," in *Network Operations and Management Symposium (NOMS), 2010 IEEE*. IEEE, 2010, pp. 286–292.

[33] M. F. F. d. Carmo, J. E. B. Maia, G. Siqueira *et al.*, "An internet traffic classification methodology based on statistical discriminators," in *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*. IEEE, 2008, pp. 907–910.

[34] F. Lisi and D. Malerba, "Inducing multi-level association rules from multiple relations," *Machine Learning*, vol. 55, no. 2, pp. 175–210, 2004.

[35] R. J. G. B. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Advances in Knowledge Discovery and Data Mining, 17th Pacific-Asia Conference, PAKDD 2013, Gold Coast, Australia, April 14-17, 2013, Proceedings, Part II*, 2013, pp. 160–172.

[36] R. J. G. B. Campello, D. Moulavi, A. Zimek, and J. Sander, "Hierarchical density estimates for data clustering, visualization, and outlier detection," *TKDD*, vol. 10, no. 1, p. 5, 2015.

[37] G. Buehrer, R. L. de Oliveira Jr., D. Fuhry, and S. Parthasarathy, "Towards a parameter-free and parallel itemset mining algorithm in linearithmic time," in *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, 2015, pp. 1071–1082.

[38] Y. He, H. Tan, W. Luo, S. Feng, and J. Fan, "MR-DBSCAN: a scalable mapreduce-based DBSCAN algorithm for heavily skewed data," *Frontiers of Computer Science*, vol. 8, no. 1, pp. 83–99, 2014.

[39] S. Moens, E. Aksehirli, and B. Goethals, "Frequent itemset mining for big data," in *Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA*, 2013, pp. 111–118.

[40] B. Panda, J. Herbach, S. Basu, and R. J. Bayardo, "PLANET: massively parallel learning of tree ensembles with mapreduce," *PVLDB*, vol. 2, no. 2, pp. 1426–1437, 2009.

[41] D. Dahiphale, R. Karve, A. V. Vasilakos, H. Liu, Z. Yu, A. Chhajer, J. Wang, and C. Wang, "An advanced mapreduce: Cloud mapreduce, enhancements and applications," *IEEE Transactions on Network and Service Management*, vol. 11, no. 1, pp. 101–115, 2014.

[42] A. Spark, "The Apache Spark scalable machine learning library. Available: https://spark.apache.org/mllib/," 2015.

[43] V. K. Bumgardner and V. W. Marek, "Scalable hybrid stream and hadoop network analysis system," in *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*. ACM, 2014, pp. 219–224.

[44] Y. Jeong, "Big Telco Real-Time Network Analytics Available: https://spark-summit.org/eu-2015/events/big-telco-real-time-network-analytics/," *Spark summit, Amsterdam, Netherland, October 27-29*, 2015.

[45] K. Swetha, S. Sathyadevan, and P. Bilna, "Network data analysis using spark," in *Software Engineering in Intelligent Systems*. Springer, 2015, pp. 253–259.

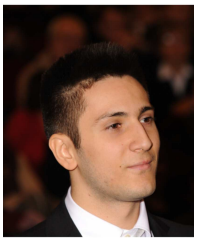
[46] D. Apiletti, E. Baralis, T. Cerquitelli, P. Garza, and L. Venturini, "SaFeNeC: a Scalable and Flexible system for Network data Characterization," in *NOMS*, 2016.

Daniele Apiletti is an assistant researcher of the Database and Data Mining Group at the Dipartimento di Automatica e Informatica of the Politecnico di Torino since January 2009. He holds a Master degree (2005) and a PhD (2008) in Computer Engineering from Politecnico di Torino. He is also adjunct professor for the Database Management Systems course at Politecnico di Torino, and Chief Data Scientist at Ooros (Turin, Italy). His research interests are in the field of NoSQL databases, large-scale data mining techniques, and Big Data machine learning, with specific focus on network-traffic, sensor-data, and social-business distributed analyses.

Elena Baralis has been a full professor at the Dipartimento di Automatica e Informatica of the Politecnico di Torino since January 2005. She holds a Master degree in Electrical Engineering and a Ph.D. in Computer Engineering, both from Politecnico di Torino. Her current research interests are in the field of database systems and data mining, more specifically on mining algorithms for very large databases and sensor/stream data analysis. She has published over 100 papers in international journals and conference proceedings. She has served on the program committees or as area chair of several international conferences and workshops, among which VLDB, IEEE ICDM, ACM SAC, DaWak, ACM CIKM, PKDD.

Tania Cerquitelli has been an assistant professor at the Dipartimento di Automatica e Informatica of the Politecnico di Torino, Italy, since October 2011. Her research interests include self-learning methodologies, the design of innovative algorithms to perform large-scale data mining, novel and efficient data mining techniques for IoT applications. Tania has published more than 70 scientific publications and has served as referee for many international journals. She has been involved in many research projects addressing different topics (e.g., energy efficiency, network traffic analysis, IoT) in the data mining research area. Tania got the master degree in Computer Engineering (in 2003) and the PhD degree (in 2007) from the Politecnico di Torino, Italy, and the master degree in Computer Science (in 2003) from the Universidad De Las Amricas Puebla.

Paolo Garza received the master's and PhD degrees in computer engineering from the Politecnico di Torino. He has been an assistant professor (with non-tenure track position) at the Dipartimento di Automatica e Informatica, Politecnico di Torino, since December 2010. His current research interests are in the fields of data mining, database systems, and big data analytics. He has worked on the classification of structured and unstructured data, clustering, and itemset mining algorithms.



Danilo Giordano received his M.Sc. Degree in Computer Engineering from Politecnico di Torino, Italy, in 2013. In 2014, he joined the Telecommunication Networks Group of Politecnico di Torino as Ph.D.candidate. During Summer 2014, he was a research intern at Narus Inc., now part of Symantec, to work on anomaly detection techniques. During the first 6 months of 2016, he was a visiting student at CAIDA research centre to extend the BGPStream tool, and to perform Big Data analysis of BGP traffic by using Apache Spark. His research interests cover several aspects of network traffic characterization, monitoring, anomaly detection, Big Data, and security on the Internet.

Marco Mellia (SM'08) graduated from the Politecnico di Torino with Ph.D. in Electronic and Telecommunication Engineering in 2001, where he held a position as Associate Professor. In 2002 he visited the Sprint Advanced Technology Laboratories working at the IP Monitoring Project (IPMON). In 2011, 2012, 2013 he collaborated with Narus Inc, CA, working on traffic monitoring and cyber-security system design. Since 2015 he is collaborating with Cisco Systems for the design of cloud monitoring platforms. He has co-authored over 250 papers published in international journals and presented in leading international conferences, all of them in the area of communication networks. He won the IRTF ANR Prize at IETF-88, and best paper award at IEEE P2P'12, ACM CoNEXT'13, IEEE ICDCS'15. He participated in the program committees of several conferences including ACM SIGCOMM, ACM CoNEXT, ACM IMC, IEEE Infocom, IEEE Globecom and IEEE ICC. He is Area Editor of ACM CCR, ACM/IEEE Transactions on Networking, and IEEE Transactions on Network and Service Management. His research interests are in area of traffic monitoring and big data analysis, with applications to traffic classification, management and security.

Luca Venturini is a PhD student at Politecnico di Torino. He holds a master's degree from both Telecom Paristech, France, and Politecnico di Torino, Italy. While still studying, he spent several months in research institutes as Bell Labs, CERN and EURECOM. His current research interests are in applications of machine learning on societal issues and very large datasets.