Online anomaly detection leveraging stream-based clustering and real-time telemetry

Andrian Putina[1], Dario Rossi[2]
[1]Telecom Paris, [2]Huawei Technologies France
andrian.putina@telecom-paris.fr, dario.rossi@huawei.com

***Abstract*—Recent technology evolution allows network equipment to continuously stream a wealth of "telemetry" information, which pertains to multiple protocols and layers of the stack, at a very fine spatial-grain and high-frequency. This deluge of telemetry data clearly offers new opportunities for network control and troubleshooting, but also poses a serious challenge for what concerns its real-time processing. We tackle this challenge by applying *streaming machine-learning* techniques to the continuous flow of control and data-plane telemetry data, with the purpose of real-time detection of anomalies. In particular, we implement an anomaly detection engine that leverages DenStream, an unsupervised clustering technique, and apply it to features collected from a large-scale testbed comprising tens of routers traversed up to 3 Terabit/s worth of real application traffic. We contrast DenStream with offline algorithms such as DBScan and Local Outlier Factor (LOF), as well as online algorithms such as the windowed version of DBScan, ExactSTORM, Continuous Outlier Detection (COD) and Robust Random Cut Forest (RRCF). Our experimental campaign compares these seven algorithms under both accuracy and computational complexity viewpoints: results testify that DenStream (i) achieves detection results on par with RRCF, the best performing algorithm and (ii) is significantly faster than other approaches, notably over two orders of magnitude faster than RRCF. In spirit with the recent trend toward reproducibility of results, we make our code available as open source to the scientific community.**

***Index Terms*—Anomaly detection algorithms; Stream learning; Machine learning; Model driven telemetry; Network monitoring and measurements**

I. Introduction

Nowadays network Operations and Management (OAM) increasingly relies on the ability to stream and process, in near real-time, useful "features" from network equipment. An integral part of the OAM task is, e.g., to ascertain whether the operational conditions are *normal* or *anomalous* and intervene, when needed, by quickly repairing eventual problems.

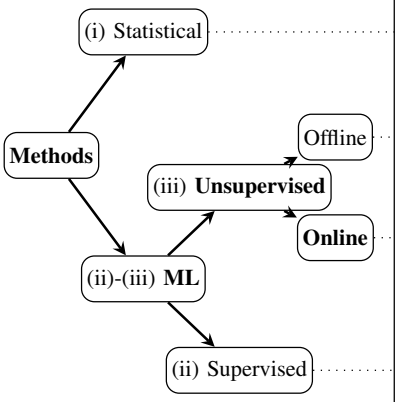
Simple Network Management Protocol (SNMP) has long been the de facto standard to gather fairly coarse information from the network management, control and data planes. Consequently, SNMP has been used for anomaly detection for long time [1]. In the SNMP paradigm, the server initiates the data collection from hundreds of devices, with a pull-based approach, at traditionally low frequency (i.e., in the order of minutes). More recently, Model-driven telemetry (MDT) [2]–[5] has emerged as an interesting alternative to SNMP: instead of having to periodically poll at a low rate (as in SNMP), under MDT subscribers receive continuous stream of operating state information in a standard structured format. In addition to supporting periodic export, MDT further enables to trigger data publication when specific conditions are met.

Rather typically, a common workflow to several vendors (such as Cisco [3], Arista [4] and Huawei [5]) is to express features via Yet Another Next Gen (YANG) [6], [7] data models, encoded with the Google Protocol Buffer (GPB) format, that are then transmitted via the Google Remote Procedure Call (GRPC) protocol. While the use of standard formats and protocol for their export is very desirable, and while the abundance of information is desirable for fine-grained monitoring, it becomes necessary to also process MDT data *as it is streamed* – a challenging task at the heart of our work. First and foremost, under MDT the data must be processed *in real-time*, which puts a practical cap on the algorithmic complexity. Second, as the data *is streamed continuously*, no assumption on data distributions or length can be made a priori. Third, *data cannot be stored* and algorithms have to perform a single pass on it, which significantly limits the algorithmic design space.

Under these requirements, it is crucial to extract, with as simple operations as possible, as much information as possible from the incoming measurements, as these have to be discarded immediately after processing. However, while the *data science community* (e.g. IEEE Computational Intelligence Society or ACM Special Interest Group on Knowledge Discovery and Data Mining[1]) flourish with *stream-based* algorithms (e.g. where models are built and maintained incrementally), these are seldom used by the *network community* (e.g., IEEE Communication Society or ACM Special Interest Group on Data Communication[2]). For example, although anomaly detection is surely not a green field, stream-based approaches are not popular yet: we argue that that streaming network telemetry is a perfect use case for stream-mode machine learning.

In this work, we present an anomaly detection engine, called Online anomaly detection in Data Streams (ODS), that leverages the DenStream [29] unsupervised clustering algorithm. As example of application, we consider Content Service Provider (CSP) datacenters that, following recent trends [30], are designed with Border Gateway Protocol (BGP) as the only routing protocol and from which the datasets are streamed and collected as telemetry timeseries (at a fast sampling rate of 5s). The dataset is gathered from a large scale CSP testbed where

[1]http://www.ieee-cis.org - https://www.kdd.org
[2]https://www.comsoc.org - https://www.sigcomm.org/



Y = Yes; N = No; CBA = Collected by authors from known sources; MLA = Manually labeled by authors; LOT = Labeled by online tools

Domain	Method	Dataset	Available	Labels	Reference
Backbone networks	PCA	Private	N	MLA	Lakhina et al. [8]
BGP anomalies	PCA	CBA	N	MLA	Huang et al. [9]
	GLRT	CBA	N	MLA	Deshpande et al. [10]
	t-test	CBA	N	MLA	Ganiz et al. [11]
Flood attacks	Cov.	KDD'99	Y	Y	Yeung et al. [12]
Intrusion detection	PCC	KDD'99	Y	Y	Shyu et al. [13]
Wireless sensor networks	kNN	IBRL/GDI	Y/Y	MLA/MLA	Rajasegarar et al. [14]
Traffic classification	K-Means	Testbed/CBA	N/N	MLA	Munz et al. [15]
	DBScan	CBA	N	LOT	Mazel et al. [16]
	DBScan	KDD'99/CBA/CBA	Y/N/N	Y/LOT/Y	Casas Hernandez et al. [17]
	LOF	DARPA	Y	Y	Lazarevic et al. [18]
Twitter spam	DenStream	CBA	N	Y	Miller et al. [19]
Traffic classification	iGDCA	CBA/CBA	N/N	N/Y	Dromard et al. [20]
	DenStream	DARPA	Y	Y	Miller et al. [21]
Intrusion detection	SVM [22]	NSL-KDD	Y	Y	Wang et al. [23]
	Neural Network	KDD'99	Y	Y	Subba et al. [24]
	Neural network	KDD'99	Y	Y	Poojitha et al. [25]
BGP Anomalies	Decision Tree (C4.5)	CBA	N	MLA	Li et al. [26]
	SVM	CBA	N	MLA	de Urbina Cazenave et al. [27]
	SVM	CBA	N	MLA	Al-Rousan and Trajkovic [28]

Fig. 1. Summary of the State of the Art in Anomaly Detection applied to Computer Networks. Related work is grouped in three branches depending on the underlying algorithmic family: *Statistical*, *Supervised* and *Unsupervised* (the latter branch, that is closest to this work, further subdivided in *Offline* vs *Online* techniques). For each work we point out the *method*, the *dataset* along with its *public availability* and *labeling* characteristics.

BGP anomalies (as categorized in [31], direct unintended BGP anomalies, misconfigurations causing leaks, link failures, blackouts or cable break) are injected in a controlled fashion. Collected features are exported via YANG telemetry, which makes them more easily identifiable and possibly portable across vendors. The set of hundreds of continuously streamed features compose the multivariate time-series to which our proposed online algorithm is applied. Summarizing our main contributions:

- We devise Online anomaly detection in Data Streams (ODS), a stream-mode anomaly detection algorithm apt at operating over telemetry data, offering its open-source implementation at [32].
- We perform an exhaustive evaluation over the available datasets, comparing ODS with classic offline (e.g., DBScan [33], Local Outlier Factor [34]) and online approaches (the windowed variant of DBScan [33], Robust Random Cut Forest [35], ExactStorm [36] and Continuous Outlier Detection [37]): results show ODS to be not only, by far, the fastest algorithm, but also to be among the most reliable ones across multiple information retrieval metrics.

The rest of this article is organized as follows. We first survey the related work and provide background material on the algorithms (Sec. II). We next describe the testbed and publicly available dataset (Sec. III). We use part of the dataset to conduct hyperparameter calibration of all the considered algorithms (Sec. IV), and to avoid overfitting, gather experimental results on the rest of the dataset (Sec. V). Finally, we summarize our findings and discuss the perspective they open (Sec. VI–VII): in a nutshell, our results show that stream mode algorithms such as ODS are able to accurately process telemetry data in real-time for a very limited computational cost, making them directly deployable in routers.

II. Related Work

This section overviews related work focused on outlier detection in computer networks. In particular, Sec. II-A presents a taxonomy of the relevant work, illustrated in Fig.1, from the network domain viewpoint, whereas Sec. II-B introduces background material concerning the unsupervised clustering techniques that are relevant for our work.

A. Outlier detection in computer networks

Hawkins [38], defines an outlier as "*an observation, which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism.*" Generally, anomalies are categorized as point, contextual or collective outliers: *Point outlier* is an object that significantly deviates from all the objects in the dataset; *Contextual outlier* is an object that significantly deviates from the objects in a context (eg. period in a timeseries); *Collective outlier* is a subset of objects that significantly deviates from the entire dataset (i.e., individual objects of the subset might not be outliers but all of them, together, constitute a collective outlier). The events we consider in this work (see Sec. III) belong to the point and collective outlier classes. In several domains (fraud/intrusion detection, public health etc.) it is assumed that the number of anomalous objects is (much) smaller than the normal objects. For this reason, several methods generate a *normal* (baseline) model of the system and label *anomalous* all the objects that are significantly different. As illustrated in Fig.1, these methods fall into three main groups, namely *(i) statistical*, *(ii) supervised* and *(iii) unsupervised* learning.

(i) Statistical methods use probabilistic models to detect changes in the data. Principal Component Analysis (PCA) is used in [8] to detect anomalous traffic volume in backbone networks by reducing the n-space of variables into a k-subspace corresponding to the *normal* behavior and a m-subspace corresponding to *anomalies* and *noise*. The subspace method is used in [9] to detect BGP anomalies, extracting the

amount of update messages from raw BGP updates every 10 minutes and processing the data in batches of 200 samples. Similarly, [10] and [11] extract features from raw BGP updates (BGP volume, Autonomous System (AS)-Path, etc.) every 5 minutes and perform anomaly detection using the Generalized Likelihood Ratio Test (GLRT) and the t-test respectively. Finally [12] contrasts the covariance matrix of *n* objects against a *normal* precomputed covariance matrix to detect flooding attacks in the *KDD'99* [39] dataset while [13], using the same dataset, proposes a Principal Component Classifier (PCC) yielding an anomaly score for both major and minor subspace components. These methods are interesting but inherently obscure for the human operator that needs to interpret the results, as the semantic of the original feature domain is lost due to the projection transformation – a significant matter of concern that render the techniques less appealing for practical purposes [40], [41].

(ii) Supervised methods learn both *normal* and *anomalous* behaviors from *labeled data* and then classify each new object *normal* or *anomalous* depending on which class fits to. Due to the lack of data, most of the methods use *KDD'99* and *NSL-KDD* [42] datasets or have to manually label private datasets (MLA in Fig.1). Among the most used algorithms there are Decision Trees (C4.5) [43], Support Vector Machines (SVM) [22] and Artificial Neural Networks (ANN) [44]. Authors of [23] obtains a $2\times$ SVM training time reduction using the *NSL-KDD* dataset augmented and transformed by the logarithm marginal density transformation while [25] and [24] use the *KDD'99* dataset to evaluate their ANN models which reduce false positive rate and minimize overall computational overhead respectively. Regarding BGP events, [26] trains and tests a decision tree (C4.5) using features extracted every minute from RouteViews and RIPE NCC archive during known misconfiguration events, Slammer worms and electricity blackouts. The same source, sampled every 30 seconds, is used by [27] which compares different algorithms as decision trees, Naive Bayes and SVM. They process the data using a sliding window of 10 samples, corresponding to 5 minutes, which slides of two minutes every time. Their system raise an alarm if at least 60% of the samples in the window are labeled anomalous allowing them to detect events as early as four minutes. The importance of BGP features is studied in [28] applying both Fisher [45] and mRMR [46] feature selection techniques, which concludes that 65% of the selected features are *volume-based* (i.e. BGP announcements, IGP packets, EGP packets etc.) and that these are more relevant and perform better than *AS-path* features. Supervised approaches are however of little portability across datasets, where features and labels differ, and the applicability of such models is therefore limited to the very specific use-case under study.

(iii) Unsupervised methods could prove particularly useful to detect outliers as they are able to find unknown patterns without using labeled data: outliers are identified as items different from the previously found patterns. Best known approaches [14]–[21], [47] uses distance (or density) to group together similar objects and label *anomalous* those far from the neighbors, and can be further categorized into *offline* vs *online* methods. Offline algorithms (such as K-Means [48], DBScan [33], Local Outlier Factor (LOF) [34]), require the access to the *entire dataset at once* (to compute centroids-objects distances or pairwise distances) and iteratively converge to a final solution. Online algorithms (such as iGDCA [49] and DenStream [29]) are instead designed to build, maintain and update models incrementally *at each new sample*. The above classes of work are closer to ours and deserve a deeper look.

Offline unsupervised methods are used in [14]–[18] for outlier detection. For instance, [14] applies k-NN, a distance-based method to detect anomalies in wireless sensor networks, whereas [15] uses K-Means to detect anomalous flows (i.e. counters of bytes, packets etc.). Several algorithms (i.e. unsupervised SVM, LOF, k-NN) are instead compared on the DARPA dataset in [18], asserting that the best performing one is LOF – which we thus include in the comparison. Density-based sub-space clustering methods are used in [16], combining evidence accumulation to identify anomalies. To reduce the high computational complexity of such offline methods, [17] utilizes a discrete time-sliding window to extract and aggregate different flow-resolution levels, in time slots of fixed length ΔT – a reasonable compromise approach that we also consider in this work.

Online unsupervised algorithms have been used more rarely for network anomaly detection [19]–[21], [47]. Authors in [20] employ a discrete time-sliding window and an incremental grid clustering algorithm to detect anomaly traffic in the core network of a Spanish Internet Service Provider (ISP). Closer to our work are [19], [21] that employ DenStream at the application and network layers respectively. In particular, [19] use DenStream to successfully detect Twitter spam using a (tiny) dataset containing approximately 3000 normal and 200 manually labeled spam entries. Authors in [21] cluster normal vs anomalous packets in the DARPA dataset operating in the data plane, and directly leverage *packet payload*, using the numerical value of each byte HTTP payloads as input features. As such, both domains of application in [19], [21] are rather far from the control-plane telemetry use-case of this paper, and neither [19] nor [21] carry on a systematic evaluation of multiple algorithms as we do in this work.

Additionally, from a practical viewpoint [21] requires continuous hyperparameter tuning and furthermore assuming *prior knowledge* of the percentage of anomalous packets – and cannot thus be readily deployed. In contrast, we make no assumption on the data and further propose principled and automated tuning methodology – that are robust to environmental condition changes.

With respect to our own previous conference work [47], we (i) ameliorate our methodology by using a *dynamic threshold*, as well as (ii) systematically compare ODS against a large set of algorithms. In particular, concerning (i) our previous work [47] required either a minimum number K_T of sequential anomalies from the same node, or a contemporary minimum number of independent detection from K_S from several nodes. Although the two methods greatly suppress false positives, single-point in time outliers, or anomalies local to a single node, can go unnoticed as recognized in [47] – which the dynamic threshold we propose in this paper successfully avoids

(see Sec.IV-C). As for (ii), a systematic and conservative comparison of performance and complexity results shows ODS to be among the most reliable algorithms across multiple information retrieval metrics (Sec.V-B) and, by far, the fastest algorithm (Sec.V-C) in the batch, which makes it suitable for deployment.

B. Overview of clustering algorithms

We now provide background information on the clustering algorithms that we will be using as building blocks for our system in this paper. A summary of the algorithms compared in this work, is present in Table I, along with their main parameters. We point out that the ultimate goals of some of these algorithms is to perform clustering: so while this section briefly covers each algorithm, we defer to Sec. IV a more formal description of our methodology to leverage clustering output for anomaly detection purposes.

DBScan is a data clustering algorithm proposed by Ester et al. [33]. It is a density-based clustering algorithm: it computes the distances between the samples and clusters altogether the points which are neighbors (i.e. whose distance is less than ϵ). By computing the ϵ neighborhood of each point, it is able to discover clusters of arbitrary shape. The points falling in low-density regions (whose nearest neighbors are far) are labeled as *noise*. In particular, the algorithm defines the local point density p by two parameters: ϵ and $MinPts$. The first one is the radius and defines the neighborhood of a point p. The neighborhood is thus the group of all the points within the radius ϵ from the point p:

$$N_\epsilon(p) = \{q \in D | dist(p,q) \leq \epsilon\} \tag{1}$$

$MinPts$ instead defines the minimum number of samples in a given neighborhood. Combining ϵ and $MinPts$, the density of an object p is $high$ when in the $N_\epsilon(p)$ there are $MinPts$ or more and low otherwise. Points with $high$ density are called $core$ points while low density points in the neighborhood of a core point are called $border$ points. Finally low density points are outliers. The clusters are obtained grouping together density-reachable points where a point p is directly density-reachable from a point q if $p \in N_\epsilon(q)$ and $|N_\epsilon(q)| > MinPts$. The outliers are the points not belonging to any cluster C_i.

The time complexity of the algorithm is $\mathcal{O}\left(n^2\right)$ [50] reducible, using efficient indexing data structures (i.e. R^*-tree), to $\mathcal{O}\left(n \log\left(n\right)\right)$ [33] on average.

wDBScan is the windowed version of DBScan. Similar to [17], the algorithm is applied to a batch of samples of length w at a time. When a new sample is available, the window advances by one step, removing the oldest sample and adding the newest one. Thus, in addition to DBScan's ϵ and $MinPts$, it adds the window size w parameter. The complexity of this approach then becomes $\mathcal{O}\left(w\ n \log\left(w\right)\right)$.

Local Outlier Factor (LOF) is a data clustering algorithm proposed by Breunig et al. [34] to detect *anomalous* data point by measuring local densities. The algorithm computes the local density of a point as the distance of the k-th nearest neighbor. It is possible in this way to identify regions of similar densities and the points with low density are considered outliers.

TABLE I
ALGORITHMS COMPARED IN THIS WORK

Algorithm	Type	Parameters
DBScan	Offline	ϵ, *MinPTS*
LOF	Offline	*n_neighbors*, *contamination*
wDBScan	Windowed	ϵ, *MinPTS*, w
ExactSTORM	Online	R, K, w
COD	Online	R, K, w
RRCF	Online	t, w, contamination
DenStream	Online	ϵ, λ, β, μ

In particular LOF computes, for each sample, the euclidean k-distance d_k to its k-th nearest neighbor. Distances are then used to compute the reachability distance of two points p and q as the maximum of their real distance and the k-th distance of the second point.

$$reachDist_k(p,q) = max(k - dist(q), dist(p,q)) \tag{2}$$

and the local reachability distance of an point p as the average reachability distance of p from its neighbors:

$$lrd_k(q) = {}^1/\left[\frac{\sum_{q \in N_K(p)} reachDist_k(p,q)}{|N_k(p)|}\right] \tag{3}$$

In the end, the local outlier factor of an object is the average local reachability distance of the neighbors divided by the object's own local reachability density, i.e.:

$$LOF_k(p) = \frac{\sum_{q \in N_k(p)} \frac{lrd(q)}{lrd(p)}}{|N_k(p)|} \tag{4}$$

Objects with similar density will have a LOF value close to 1, higher density objects (*normal*) will have LOF lower than 1 while a LOF value higher than 1 will mean an object which has a lower density than the neighbors and thus an *outlier*. The decision function can be properly tuned by observing the scores of the top-most abnormal samples (*contamination*) in the dataset. LOF hyperparameters are $n\_neighbors$ and *contamination* and its time complexity is $\mathcal{O}\left(n \log\left(n\right)\right)$ [34].

DenStream is a clustering algorithm proposed by Cao et al. [29]. It is an algorithm designed for data streams, which extends the density-based strategy introduced in DBScan making it viable for online model construction. First of all, the algorithm uses a damped window model to weight the samples: older ones become less important than newer ones via a fading function

$$f(t) = 2^{-\lambda t}, \lambda > 0 \tag{5}$$

where λ is the aging parameter. The main idea of the algorithm is the introduction of the so called *micro-clusters (mc)*, i.e., group of close points $p_{i_1}, ..., p_{i_n}$ with creation time stamps $T_{i_1}, ..., T_{i_n}$. A *mc* is defined as a (w, c, r) where *w* is the weight, *c* is the center and *r* is the radius of the *mc*. The weight

w is given by the number of elements in the *mc* weighed by their generation time T_{ij} with respect to the current time t:

$$w = \sum_{j}^{n} f(t - T_{i_j}) \tag{6}$$

Similarly,

$$c = \frac{1}{w} \sum_{j}^{n} f(t - T_{i_j}) p_{i_j} \tag{7}$$

$$r = \frac{1}{w} \sum_{j}^{n} f(t - T_{i_j}) dist(c, p_{i_j}) \tag{8}$$

where $dist(c, p_{i_j})$ is the euclidean distance between point p_{i_j} and the center c. By breaking clusters into *mcs*, DenStream allows to dynamically construct clusters of arbitrary shapes. The *mc* weight *w* plays a key role in the model construction, as it discriminates between *outlier* ($w < \beta\mu$) vs *core* ($w > \beta\mu$) micro-clusters (where β and μ are free parameters).

When a new sample is available, DenStream (i) merges it to the nearest *core mc* provided that the radius of the merged cluster does not exceed a given threshold ϵ; otherwise, DenStream (ii) attempts to merge the point to the closest *outlier mc*, and (iii) a new outlier *mc* is finally created by the point if the merge fails.

Not only *mcs* are easy to maintain *incrementally* at each new data point, but notice that model construction is a continuous process in DenStream: an outlier *mc* can indeed become a core *mc* when its weight increases as new points are added to it. Similarly a core *mc* becomes an outlier *mc* (and ultimately vanishes) if no new data points are added for long periods. The authors show that the minimal time span for a core *mc* fading into an outlier one is $Tp = \frac{1}{\lambda} log(\frac{\beta\mu}{\beta\mu - 1})$ therefore it is natural to check them every *Tp* time periods. Two *offline* phases can be found in DenStream: *Initialization* and *Generating Final Clusters*. The authors propose indeed to obtain the initial *mc* as the output of DBScan applied to the first *InitN* points, called *buffer*, and then maintain them incrementally. Similarly, they propose to obtain the *final clusters*, when requested, applying again DBScan to the set of core *mcs* considering them as a virtual point located at the center of the *mc*.

ExactSTORM is an online anomaly detection method proposed by Angiulli et al. [36]. The method uses a sliding window model and stores the data instances in nodes of a suitable data structure called Indexed Stream Buffer (ISB). Each node contains the data instance *p*, its arrival time *p.t*, the number of succeeding neighbors *p.count_after* and the list, of size at most *k*, of the preceding neighbors *p.nn_before*. The ISB data structure supports the *range query search*, that given an object *p* and a real number $R > 0$, returns the objects in the ISB whose distance to *p* is not greater than R. For each incoming instance *p*, a range query is performed in ISB which returns the list of its preceding neighbors *Q*. For each $q \in Q$, the number of succeeding neighbors *q.count_after* is increased by 1. Finally, *p* is inserted into the ISB while the expiring instance *o* is removed from the data structure.

All the operations previously described are performed by the so called *Stream Manager* which updates the ISB for each incoming instance. It is subsequently up to the *Query Manager* to scan the ISB searching for outliers.

For each instance *p* in the ISB, the *Query Manager* discriminates between *inliers* and *outliers* by considering the sum of the succeeding neighbors *p.count_after* with the size of the preceding neighbors list *p.nn_before*. If the sum is lower than *k*, then *p* is an outlier and inlier otherwise.

Continuous Outlier Detection (COD) is an online anomaly detection method proposed by Kontaki et al. [37]. Similar to [36], the method uses a sliding window model and stores the instances in a data structure that supports range queries efficiently (e.g. M-Tree). The authors observe that (i) a departing instance can transform inliers into outliers, (ii) an incoming instance can transform outliers into inliers and (iii) not all the instances are affected by the expiring ones. Since only the neighbors of the expiring instances have to be updated, COD uses a priority queue (Fibonacci queue) to schedule processing of affected instances. The method stores for each instance of the stream *p*, its arrival time *p.time*, its expiration time *p.exp*, the list of its preceding neighbors *p.P* and the number of the succeeding neighbors $p.n^+$.

When a new instance *p* is available, the algorithm sets the expiration time to w samples from the current time. It subsequently performs a query which returns the list of objects (*p.P*) lying at distance at most *R* from *p*. Comparing the total number of neighbors against the threshold *k*, the algorithm discriminates inliers from outliers. If *p* is an outlier it is added to the outlier list; it is added to the inlier list and to the priority queue otherwise. The key in the priority queue is set according to the minimum expiration time of all its preceding neighbors *p.P*. For each instance $q \in p.P$, the number of succeeding neighbors $q.n^+$ is incremented by 1. When the total number of neighbors exceeds *k*, *q* is promoted to the inliers list and added to the priority queue. The key in the priority queue follows the minimum neighbors expiration time previously described.

When an instance *o* expires, it is removed from the range query data structure and from the preceding neighbors lists. To do so, the priority queue is polled until all the elements set to be checked in the current time are extracted. Each extracted element *q* is either added to the outlier list if its total number of neighbors falls below *k*, or its key is updated and q is reinserted into the queue otherwise.

Robust Random Cut Forest (RRCF) is an online anomaly detection method proposed by Guha et al. [35]. It is an ensemble algorithm composed by *t* different models which maintains *w* instances (tree size) in binary trees. Each instance $p \in w$ is isolated in a leaf of the tree while internal nodes act as splitting nodes. Each internal node, in addition to splitting criterion (attribute and value) maintains also the dimensions bounding box (support) of all the instances in the sub-tree.

Given a set of instances *S* and a tree *T(S)*, when a new instance *p* is available, the algorithm tries to insert it from the root. It first sums together the supports of all the dimensions contained in the bounding box and extracts a random number $r \in [0, \sum_i (x_i^h - x_i^l)]$ which determines the attribute and the splitting value. If the split separates the instance *p* from the

remaining tree, the algorithm generates a new splitting node with a branch containing p and another one containing the previous tree *T(S)*. If the split does not isolate *p*, then *p* follows the path of the existing tree and the procedure is repeated on each sub-tree until the instance is isolated. All the bounding boxes on the path of p are updated.

When an instance o departs, the algorithm removes its parent and replaces it with the sibling. All the bounding boxes starting from o upwards are updated. Notice that all the operations previously described (insertion, deletion, bounding box updating process, etc.) are repeated for each of the t trees in the ensemble.

The insertion and removal of each instance in the tree leads to a modification of the tree structures. The variation in tree complexity is used to determine the anomaly score. Given a set of points Z, a point p and a tree T, the depth of p is $f(p, Z, T)$. Assigning to each left branch of the tree the bit 0 and the right branches the bit 1, the model complexity $|M(T)| = \sum_{p \in Z} f(p, Z, T)$ is the number of bits required to write down the description of all points p in the tree. The bit-displacement of an instance x is:

$$Disp(x, Z) = \sum_{T, p \in Z - x} Pr[T] \left(f(p, Z, T) - f(p, Z - x, T') \right) \tag{9}$$

the increase in the model complexity of all other points, i.e., for a set Z, to capture the externality introduced by x, where $T' = T(Z - x)$. To avoid outlier masking (phenomenon in which dense outliers mask each other) instead of removing just x, they propose to remove a set C with $x \in C$ and obtain so the *Collusive Displacement* $CoDisp(x, Z, S)$

$$\mathop{\mathbb{E}}_{S \subseteq Z, T} \left[\max_{x \in C \subseteq S} \frac{1}{|C|} \sum_{p \in S - C} \left(f(p, S, T) - f(p, S - C, T'') \right) \right] \tag{10}$$

Anomalies correspond to large $CoDisp$ values: similarly to LOF, we properly tune the decision function by observing the scores of the top-most abnormal samples (contamination) in the dataset.

III. Testbed and Datasets

We study and compare the proposed method using publicly available datasets [51], gathered and released for the previous conference version of this paper [47]. The datasets have been collected in a state of the art testbed, comprising tens of real routers, running real protocols and traversed by Tbps traffic (Sec.III-A). The testbed is used in several experiments where anomalous events are injected in randomly chosen nodes in a controlled fashion (Sec.III-B). In turn, these controlled anomalies affect the stream of telemetry features (Sec.III-C) , as we illustrate for the sake of clarity (Sec.III-D).

A. Testbed

The dataset is extracted from a testbed replicating a traditional clos topology of a CSP datacenter shown in Fig.2. For redundancy, each leaf is connected to each spine via 4×100Gbps fiber links, so that the nodes have 25 interfaces on average. On the operational level, the datacenter is designed with BGP as the only routing protocol, following guidelines in [30].

TABLE II
Experimental datasets available at [51]

Experiment ID	Traffic Load	No. Anomalies	Duration	Used for
2	1 Tbps	11	1 h	Tuning parameters (Sec. IV)
3	1 Tbps	8	0.55 h	Tuning parameters (Sec. IV)
5	1 Tbps	12	2 h	Test parameters (Sec. V)
9	2.9 Tbps	5	0.75 h	Test parameteres (Sec. V)
10	2 Tbps	5	0.55 h	Test parameters (Sec. V)

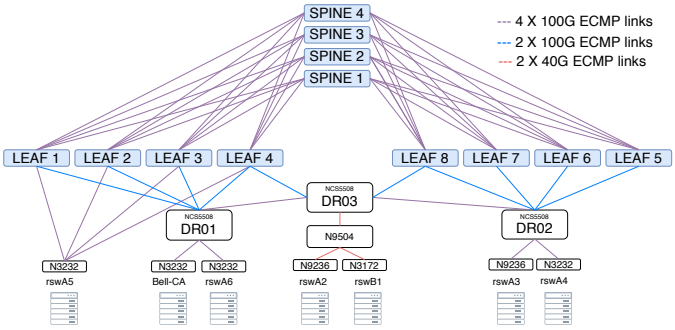


Fig. 2. Testbed replicating a traditional clos topology of a CSP datacenter

Though the testbed does not involve real users, it does use real equipment, protocols and applications typical of production networks. We thus disregard experiments collected under no traffic load (mostly useful for testing) and limitedly consider those where real application mixtures are generated from servers in the racks connected to the ToR switches (the Nexus 2/3/5000 and 9000 series) to generate up to 3 Tbps of aggregated traffic (a mixture of TCP, AMR-WP VoIP and G.711a calls, Skype-1050P, Blue Ray and 4K YouTube streaming).

B. Data collection

Multiple experiments, listed in Table II, with different characteristics and scenarios are performed. While minute-level telemetry collection is generally practice in the industry, the dataset we use in this work has been gathered with the fastest sampling period supported by the products, namely of ΔT=5 seconds. Every ΔT, each of the N nodes stream a snapshot of its F features to the collector: each experiment is a point $X \in \mathbb{R}^{NSF}$ where N is the number of nodes, S is the number of collected samples during that experiment and F the number of features. All the available features are described by YANG models [52] and then extracted, decoded and stored by Pipeline [53] as compressed CSV files. The dataset was already made available at [51] as contribution of our previous conference version of this paper. The repository [51] contains experiments undergoing different traffic *load* (from 0 to 3 Tbps), different *number* of anomalies injected (from 0 to few 10s) and different anomalies *types*, such as BGP port flapping (Link Failure - Point anomalies), BGP leaks and

BGP clears (Direct Unintended BGP Anomalies - Collective Anomalies). An interesting point is how to ensure that the synthetic anomaly injection process yields to outliers that retain similarities with anomalies found in the real-world. While a systematic quantitative evaluation of the anomaly injection process is outside the scope of this paper, it is possible to provide preliminary qualitative insights on this point. Notice that these synthetically injected anomalies represent the type of BGP anomalies that are typically found in real-data [54]; additionally, such anomalies are injected by actioning on the protocol (e.g., automatically activating/deactivating links for flapping, purposely misconfiguring tables for leaks, and resetting tables for clears) as it would happen in case of real connectivity problem (flapping) or "fat finger" (leaks, clears), so to trigger real protocol reaction. As such, while the temporal patterns of the anomalies are likely unrealistic (i.e., since they are periodical and more frequent that what it can be expected) by separating anomalies by a long enough time, we can ensure anomalies are roughly independent, and as such should independently trigger alarms. Overall, the synthetic injection process is expected to provide a sufficiently realistic benchmark.

The working condition of the system is classified in two categories, i.e., *normal* vs *anomalous*. The system works by default in normal mode, and each experiment starts with a normal period (lasting at least 40 samples), after which controlled anomalous events are injected at randomized node locations. Depending on the dataset, the anomalies are injected by spacing them by 300 seconds or more and all of them are tracked in a ground truth database available in the same repository. The groundtruth file includes the *root node* in which the event is injected, the *timestamp* and the *type*. We point out that we do not leverage ground truth information to build our data-driven models (i.e., as one would to in case of supervised classification), but rather use ground truth only to assess the performance of the unsupervised methods.

Whereas the start time of the anomalous event is known, the event duration is not deterministic: the even injection triggers the BGP update process, after which BGP converges to a new stable state. We discuss with product line experts to set an expected event duration: based on expert knowledge related to both protocol dynamics (i.e., the convergence process of BGP), as well as business objectives (i.e., the ability to gather actionable alarms on a timescale interpretable by humans), network experts consider an event ended 3 minutes after its injection time.

Clearly, as any threshold that can be set arbitrarily, its tuning may impact algorithmic performance evaluation: for instance, setting a very short event duration (sub-minute) would raise several events that would be wrongly counted as "false alarms" (since BGP did not converge yet in practice); conversely, setting a too long duration (e.g., larger than the interval between two consecutive injections) could lead to superposed event. Based on properties of the injection process, and on our preliminary observations of the timeseries, we concur with experts that 3 minutes is an advisable event duration for this datasets also from the viewpoint of machine learning experts.

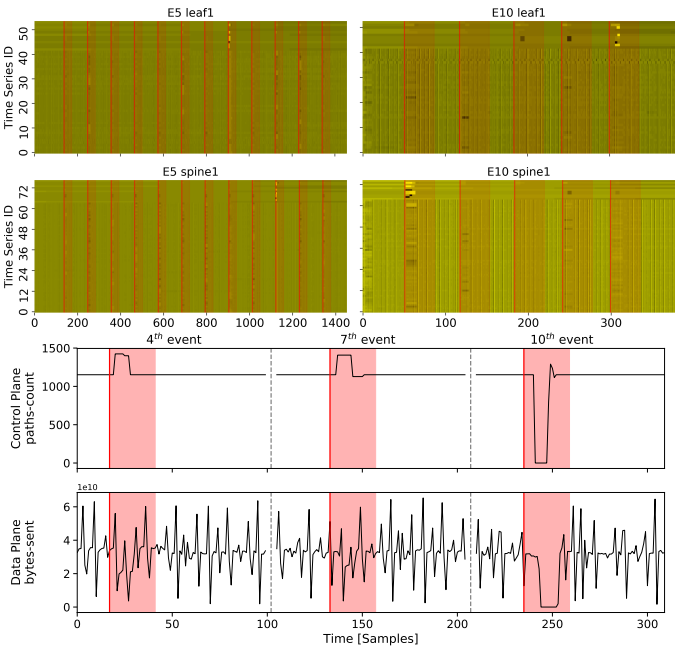


Fig. 3. Dataset at a glance: Top plots depict example of the Multivariate Time Series as heatmaps for *leaf1* and *spine1* on E5 (plots on the left) and E10 (plots on the right). Bottom plot reports temporal evolution for sample features and annotated ground truth for node *spine1* on E5: control-plane *paths-count* (top), vs data-plane *bytes-sent* on interface HundredGig0/0/0/0 (bottom).

C. Telemetry features

The streamed KPI (aka features in machine learning terms) available in the testbed are a subset of the YANG [7] state of the devices, exported by GRPC to an inbound collector. In a nutshell, YANG models define a hierarchy (i.e., tree) of data that can be used for configuration, state sharing and notifications; in the model, each node has a name, and either a *value* or a set of child nodes. At the same time, it is worth stressing that the YANG hierarchy of devices in the testbed comprises over 378,000 lines, describing a hierarchy of over 45,000 features, with *nearly 5,000 types pertaining to the BGP protocol alone*. From a machine learning viewpoint, it would be counter-productive, due to the curse of dimensionality, to apply any clustering algorithm to such a highly dimensional data. Additionally, from a network-expert viewpoint, collecting and exporting features consumes CPU and bandwidth resources: as such, it is impossible to collect, for all nodes and interfaces, the totality of the supported features – which rules out the possibility to conduct classic "feature selection" algorithms. Product line experts configured the testbed to collect the most relevant control and data plane KPIs according to their domain knowledge, and we therefore take the resulting set of features (reported in Appendix A for completeness) as a given. However, it is well known that not all features are equally important in machine learning terms: by discarding constant or categorical features from the full set of available features, we finally extract a subset of 82 non-trivial features (reported in Appendix B for completeness).

D. Dataset at a glance

For the sake of clarity, we illustrate samples of the dataset in Fig. 3, to exemplify the types of KPI signals and anomalies present in the dataset from spatial and temporal angles.

1) Spatio-temporal view: We start from a heatmap representation of the multivariate data collection in the top of Fig. 3: x-axis represent the time, y-axis represents different features whose values are encoded as colors. To portray two different datasets and nodes, we select *leaf1* and *spine1* extracted from experiments E5 and E10. It is easy to observe that in E5, all the 12 BGP clear events have a noticeable impact on *spine1* features, and to a smaller extent, on *leaf1*. While it is visually possible to distinguish the first four events in *spine1*, the same does not hold for *leaf1*: this happens since these events are injected in nodes that are directly connected with *spine1* but are at 2 hops away from *leaf1*. Thus, *leaf1* features are most noticeably affected when an event is injected in a topologically nearby node. Heatmaps on the right show that the events injected in E10 have even a less noticeable impact: link failures indeed impact only one (out of four) direct links between two nodes, and consequently only some of the features related to that particular link are affected, which can be hard to detect.

2) Temporal view: Events are more easily noticeable by considering a temporal view, on the bottom part of Fig. 3, that shows an example of CP (*paths-counts*) and DP (*bytes-sent*) features for *spine1* in E5. The ground truth is represented with a vertical red line representing the anomaly injection time and a shaded window for the anomaly duration, and we depict only 3 out of the 12 injected events for the sake of readability. From these few examples, it can be expected that accurately detecting all events, from all nodes, can be a quite challenging due to the nature of the events, that can yield to weak signals for some nodes and anomaly type.

IV. Methodology

Traditional clustering-based approaches, e.g. DBScan are mainly designed to produce clusters rather than detecting outliers or other types of anomalies, which is our ultimate goal. As such, in this section we first specify how we move from clustering to outlier detection Sec. IV-A. We next illustrate in Sec. IV-B/C the careful hyperparameter selection procedure we followed to ensure fair performance comparison in Sec. V.

A. From clustering to anomaly detection

All the considered methods succeed in making a distinction between *normal* samples (belonging to a cluster) and *outliers/noise*. We further use these peculiarities to trigger anomaly detection for each algorithm. In particular, a sample is considered *anomalous*

(i) by DBScan, if it cannot be merged to any cluster;
(ii) by wDBScan, if by adding it to the time-sliding window, it cannot be clustered together with the previous samples;
(iii) by LOF, ExactSTORM, COD and RRCF, if it is assigned an *anomalous* label
(iv) by DenStream, if it is successfully merged to an outlier-*mc*, or a new outlier-*mc* needs to be created for that sample.

Note that the exact composition and size of clusters is affected by the algorithms hyperparamers: e.g., the minimum size of a cluster is either *explicitly* (e.g., as for DBScan via $MinPTS$, cfr. Sec.IV-B) or *implicitly* specified (e.g., as for ODS, cfr. Sec.IV-C)

Algorithm 1: DenStream [and *ODS*]

1: Initialization with DBScan [skipped in ODS]
2: **while** *ns (new sample)* **do**
3: find closest core *mc* and try to merge *ns*
4: **if** $r_c < \epsilon$ $[< \bar{r} + k_r\sigma_r]$ **then**
5: merge *ns* to core *mc*
6: [return label $normal$, r_c]
7: **else**
8: find closest outlier *mc* and try to merge *ns*;
9: **if** $r_o < \epsilon$ $[\bar{r} + k_r\sigma_r]$ **then**
10: merge *ns* to outlier *mc*
11: **if** outlier *mc* weigth $> \beta\mu$ $[< \beta/(1-2^{-\lambda})]$ **then**
12: promote outlier *mc* to core *mc*
13: **end if**
14: **else**
15: generate new outlier *mc* by *ns*
16: **end if**
17: [return label $anomalous$, r_c]
18: **end if**
19: apply fading function $2^{-\lambda \cdot t}$ to all *mcs*
20: **if** t mod $T_p == 0$ **then**
21: **for** each core *mc* **do**
22: **if** $w_p < \beta\mu$ $[< \beta/(1-2^{-\lambda})]$ **then**
23: remove core *mc*
24: **end if**
25: **end for**
26: **for** each outlier *mc* **do**
27: **if** $w_o < \beta\mu$ $[< \beta/(1-2^{-\lambda})]$ **then**
28: remove outlier *mc*
29: **end if**
30: **end for**
31: **end if**
32: DBScan on core *mcs* [skipped in ODS]
33: **end while**

For the sake of illustration, we report the pseudo-code (merging, promotion and pruning phases) of the DenStream algorithm, together with the proposed changes for label assignment in ODS (emphasised and between [square brackets]) in Algorithm. 1. In particular, we propose two major changes with respect to the original version of DenStream to be found in the initialization phase and in the offline clustering part. While, in the initialization phase (line 1), the original version of DenStream uses DBScan to obtain the micro-clusters and then start maintaining them incrementally, we cluster together the first S samples of the stream assuming they are event free and not contaminated by anomalies. By doing so, we obtain a cluster of *normal* samples (normal working condition of the

TABLE III
HYPERPARAMETERS: NUMBER AND RANGE OF COMBINATIONS TESTED FOR EACH METHOD AND FINAL SELECTION

Method (num. combinations)	Parameter Name	[Range]@step	Selected Value
DBScan	ϵ	$[1, 20]$, @1	6
(500)	$MinPts$	$[2, 50]$, @2	18
LOF	$n\_neighbors$	$[1, 50]$, @1	24
(1960)	$contamination$	$[0.001, 0.2]$, @0.005	0.065
wDBScan	ϵ	$[1, 20]$, @1	9
(8000)	$MinPts$	$[2, 50]$, @2	3
	w	$[20, 100]$, @5	80
ExactSTORM	R	$[1, 20]$, @1	10
(3040)	w	$[10, 100]$, @5	95
	K	$[2, 10]$, @1	2
COD	R	$[1, 20]$, @1	12.5
(3040)	w	$[10, 100]$, @5	50
	K	$[2, 10]$, @1	5
RRCF	t	-	100
(800)	w	$[5, 100, @5$	95
	$contamination$	$[0.001, 0.2]$, @0.005	0.03
ODS	ϵ	dynamic	$\bar{r} \pm k_r \sigma_r, k_r = 3$
(135)	λ	$[0.01, 0.45]$, @0.03	0.125
	β	$[0.1, 1]$, @0.1	0.4

system). We remove, similarly, the offline part of DenStream (line 31) in which DBScan is applied to cluster together micro-clusters. Our goal is to find out if the samples, as soon as they are available, are *normal* or *anomalous*; we need so to know only if the samples can be merged to existing *normal-mc* or not. Instead, we do not need to group together different micro-clusters, which may represent different normal states, to obtain a macro cluster of *normality*. Other changes pertain to automated tuning, which we detail in Sec. IV-C.

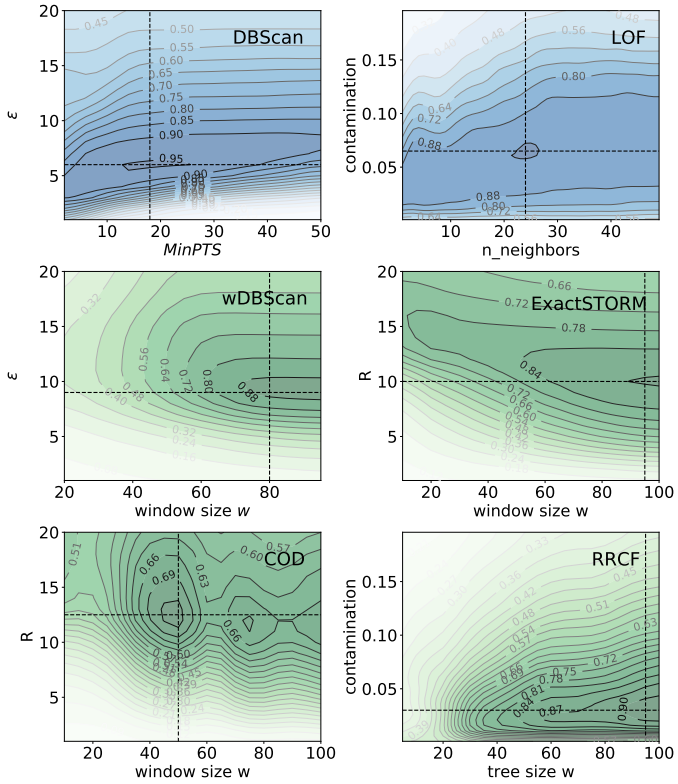


Fig. 4. Hyperparameter selection: $F_{0.5}$ heatmap for DBScan (top left), LOF (top right), wDBScan (middle left), ExactSTORM (middle right), COD (bottom left) and RRCF (bottom right). Detailed parameters in Tab. III. Selected hyperparameters at the intersection of the dashed lines.

B. Hyperparameter selection (DBScan, LOF, wDBScan, ExactSTORM, COD and RRCF)

For all the methods, we first perform a hyperparameter selection phase, to select parameters yielding to good performance as measured by classic metrics from information retrieval (i.e., precision, recall and F_β scores). Given that anomalies are rare, we use the $F_\beta = (1+\beta^2)\frac{precision \cdot recall}{\beta^2 \cdot precision + recall}$, setting $\beta = 0.5$ to account for imbalance by non-linearly interpolating precision and recall. In particular, we use E2 and E3 from Tab. II for parameter selection. Generalization capabilities of the tuned algorithms will be tested on entirely different datasets in Sec. V. The full set of parameters explored, along the total number of combination tested per protocol, and the resulting selection is summarized in Tab. III.

We use classic *grid optimization*, i.e., an exhaustive search, to find the parameters that reach the best performance. We do not perform however a blindly search, as it is of fundamental importance the research of the parameters in the correct intervals and magnitudes, therefore we follow the best practices suggested by the authors in the determination of grid boundaries.

1) Grid boundaries: For DBScan ϵ and *MinPTS* parameters, we perform a grid search varying $\epsilon \in [1, 20]$ with a unit step and $minPTS \in [2, 50]$ with step equal to 2, for a total of 500 different hyperparameters combination.

wDBscan follows the same principles for what concerns $\epsilon \in [1, 20]$ and $minPTS \in [2, 50]$. The window size *w* is searched in $w \in [20, 100]$ in steps of 5 units, which is equivalent to time windows ranging from 1 minute to 8 minutes (in line with BGP time spans), obtaining so a total of 8000 different parameters tested for each node.

The number of neighbors used by LOF is searched in $n\_neighbors \in [1, 50]$ with unit step, while we explore $contamination \in [0.001, 0.2]$ with a step equal to 0.005, obtaining a total of 1960 combinations.

For what concerns ExactSTORM and COD, we perform the *grid search* varying $R \in [1, 20]$ and $k \in [2, 10]$ with a unit step while $w \in [10, 100]$ in steps of 5 units obtaining so a total of 3040 parameters tested for each node.

The number of trees used by *RRCF* is set to $t = 100$ as it is commonly used in ensemble models. We explore the tree size $w \in [5, 100]$ in steps of 5 units while exploring $contamination \in [0.001, 0.2]$ with a step equal to 0.005, obtaining a total of 800 combinations.

2) Grid search results: Fig. 4 shows a heatmap of the $F_{0.5}$ score for DBScan (top left plot), LOF (top right plot), wDBScan (middle left plot), ExactSTORM (middle right plot), COD (bottom left plot) and RRCF (bottom right plot). Clearly, hyperparameter selection is to select the point (or points in a region) that maximizes the $F_{0.5}$ score. For each algorithm, we represent the hyperparameter space of two parameters as a heatmap, to convey an idea of the algorithm stability to (even slight) hyperparameter changes (in the region). We highlight the final hyperparameters choice directly in each plot, i.e., at intersection of the dashed lines.

We observe that DBScan performs the best for $3 < \epsilon < 8$

and $10 < MinPTS < 30$. We select $\epsilon = 6$ and $MinPTS = 18$. We also notice that $F_{0.5}$ changes quickly in ϵ, and is slowly varying in $MinPTS$.

From the top right plot, we observe that the region for which LOF produces the best results are $0.06 < contamination < 0.07$ while $20 < n\_neighbors < 30$. We select $contamination = 0.065$ and $n\_neighbors = 24$ which is close to the default parameter ($n\_neighbors = 20$).

Heatmaps in middle plots clearly show that the window size need to be $w > 70$ for both wDBScan and ExactSTORM. In the wDBScan case though, we remark that the use of a smaller time-horizon affects the $(MinPTS, \epsilon)$ heatmap so that the best selection appears to fall for $MinPTS < 6$ (much smaller that in the DBScan case) and $\epsilon \approx 9$ (slightly larger than for DBScan). We select $\epsilon = 9$, $w = 80$ and $MinPTS = 3$. We select $R = 10$, $w = 95$ and $k = 2$ for ExactSTORM.

Finally bottom plots show that $R = 12.5$, $w = 50$ and $k = 5$ are the best parameters for COD while tree size $w = 95$ and $contamination = 0.03$ are the best ones for RRCF.

C. Hyperparameter selection (ODS)

We point out that, as reported in Tab. III, the total number of hyperparameter explored is smaller (135) than in the other cases: this should provide not only a fair, but an expected conservative performance assessment of ODS performance. At the same time, since DenStream is less well known and ODS build on it, we present a more comprehensive explanation of its hyperparametrization. Particularly, we use ingenuity to:

(i) simplify hyperparameter selection by lumping factors whenever possible (as for μ^+), as well as
(ii) proposing dynamic parameterless settings based on statistical properties (for ϵ) and
(iii) resorting to grid search for the remaining ones (λ, β).

1) Maximum weight μ^+: The weight parameter μ is used jointly with the potential factor β to decide when a given outlier *mc* becomes a new core *mc* (particularly, when $w > \mu\beta$). Given the exponential fading function $f(t) = 2^{-\lambda t}$, and considering a fixed-rate sampling as in our case, the maximum weight a micro-cluster can reach is $\mu^+ = \sum_j f(j) = \frac{1}{1-2^{-\lambda}}$ (since $|f(t)| < 1$ for $\lambda > 0$) which solely depends on λ. By setting $\mu = \mu^+$ we therefore reduce the parameter cardinality, obtaining the new minimal time span for a normal cluster fading into an outlier one $T_p = \lceil \frac{1}{\lambda} log_2(\frac{1}{\beta}) \rceil$, obtained by the equation $2^{-\lambda T_p}\mu^+ = \mu^+\beta$ and the rule for outlier micro-cluster *mc* promotion becomes:

$$w > \beta/(1 - 2^{-\lambda}) \tag{11}$$

2) Radius threshold ϵ: The radius threshold is the most important parameter of the algorithm as it delimits the *anomalous* threshold. On the one hand, one may suggest a *fixed* selection of the parameter, that is to compute it as the radius of the cluster obtained merging together the data of an experiment in which no anomalies are injected (i.e. E0 and E1 - baselines). This is the approach we originally followed in [47]. At the same time, we argue that a *fixed* selection of the parameter introduces *portability* issues: as it is necessary to generate baselines for each possible combination of topology, traffic loads and BGP policies, this making the choice fragile and impractical. On the other hand, one could advocate for a *dynamic* selection of the parameter, that is automatically computed as the radius of the cluster obtained merging together the first S samples at the beginning of the model construction and keep updating the threshold as new samples are available. We follow this second path and dynamically set the radius threshold as:

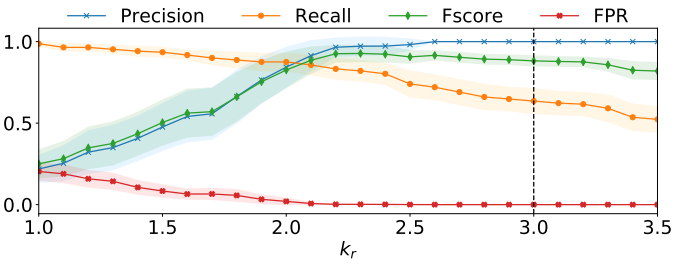


Fig. 5. ODS Hyperparameter selection: Impact of k_r for dynamic radius threshold in (12)

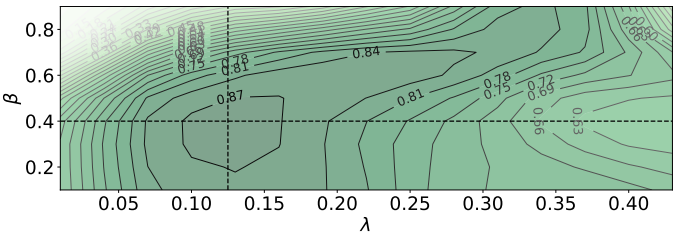


Fig. 6. ODS Hyperparameter selection: $F_{0.5}$ score for increasing fading factor λ and potential factor β applied on E2 and E3

$$\epsilon(k) = \bar{r} \pm k_r \sigma_r \tag{12}$$

where $\bar{r}$ is the incremental estimation of the radius mean while σ_r is the incremental estimation [55] of the radius standard deviation, and k_r an arbitrary parameter that allows to control (more precisely, upper bound) the false alarm rate.

We observe from (7) and (8) that the radius has a gamma type distribution (as it is the square root of sums of positives values). Without further assumptions on the radius distribution, we use Chebyshev's inequality:

$$Pr\left(|X - \mu| \geq k\sigma\right) \leq \frac{1}{k^2}, \text{ where } k > 0 \tag{13}$$

which states that for a random variable with finite expected value μ and variance σ^2, it is possible to compute a lower bound of the probability that values lie inside the interval $(\mu - k\sigma,\ \mu + k\sigma)$. For example, for $k = 3$, the interval contains at least 88.89% of the population, upper bounding false alarm rate to at most 11%. We should stress that the bound is however not tight: to confirm this, we report in Fig. 5, the false positive rate (along with precision, recall, and $F_{0.5}$) as a function of k_r, which is extremely low already for $k_r > 2$. To conservatively evaluate ODS, in the following we set $k_r = 3$.

3) Fading λ and Potential β factors: Both λ and β have a physical interpretation and play a key role in the model construction. λ is a time-related parameter that tunes the timescales at which *old samples should be considered as totally independent from the current system state*. Once λ is

fixed, the potential factor β has a geometric interpretation, as *it determines the minimum number of samples needed for outlier mc promotion to core mc*. We point out that a domain expert could be tempted to select λ and β according to physical properties of the network, whereas a machine learning expert would select λ and β as a result of data analysis and a *grid search* procedure. We adopt both viewpoints in what follows.

For instance, a network domain expert could decide to require that an outlier *mc* should have at least 3 samples before becoming a normal *mc* and set λ according to the expected convergence time of the BGP protocol. For example, choosing λ such that a sample's contribution decays 99% after 5 minutes means $2^{-\lambda \cdot 60} = 10^{-2}$ or $\lambda \approx 0.111$ while the promotion threshold, requiring at least 3 samples, is $\beta \cdot \mu^+ = \sum_{i=0}^{3-1} 2^{-\lambda i} \approx 2.78$.

We perform a grid search to question these choices, reporting in Fig. 6 the $F_{0.5}$ score for varying $\lambda \in [0.01, 0.45]$ and $\beta \in [0, 1]$. Several important takeaways can be gathered from the figure. First, the performance metrics are smoothly varying over λ and β with a fairly large region (best $F_{0.5}$ score obtained for $\lambda \in [0.10, 0.16]$ and $\beta \in [0.2, 0.5]$ approximately).

Second, performances degrades for both increasing λ and β. A too high decay factor λ leads to giving too much importance to the most recent samples while a too high β, instead, translates in a too high weight threshold. We select $\lambda = 0.125$ and $\beta = 0.4$. Notice that with $\lambda = 0.125$, $\mu^+ \approx 12$ and for $0.2 < \beta < 0.5$ the *weight* promotion threshold ranges from $2.4 < \beta \cdot \mu^+ < 6$ which translates in clusters whose weight is composed by at least 3 to 8 samples and which are in line with the expert domain choices. These findings confirm that indeed the parameters of the algorithm respect their physical interpretation and can be set accordingly.

A last remark is worth making: while we have seen that performance smoothly varies on β and λ, we point out that their selection is still primarily correlated with the telemetry sampling rate: as such, for very different sampling timescales (eg. subsecond or minutes), a new sensitivity analysis is recommended.

V. Performance evaluation

We now carefully compare the methods, using the parameters selected in the tuning phase on datasets (E2, E3), on previously unseen datasets (E5, E9, E10). We start by illustrating one example of execution of the system in Sec. V-A. We systematically compare performance of the algorithms in terms of several information retrieval metrics in Sec. V-B. We finally contrast algorithms in terms of complexity and execution times in Sec. V-C.

A. Model evolution over time

To better illustrate intrinsic differences of stream-learning vs classic algorithms, we portray the evolution of two sample models. In particular, we select ODS and the windowed version of DBScan (wDBScan) and depict their anomaly detection processes with the help of Fig. 7, using the initial portion of dataset E5 as an example. In particular, we ought to recall that whereas in the ODS case a *single model* evolve over time, in wDBScan each of the different windows yield to a *different model*. In other words, in ODS model evolution over time is smooth by design, whereas in wDBScan the similarity between outputs of models that are run on consecutive input windows, merely stem from similarity in the input data, as the time component is not otherwise explicitly exploited. In spite of the above difference, it is possible to resort to consistent visualization of the main inner state of these algorithms, such as the number of normal vs outlier clusters, their radius size and center position, etc. that are reported in Fig. 7.

In wDBScan, each time the algorithm is run, a unique increasing ID is assigned to each cluster, starting every time from 0: each sample is assigned the ID of the cluster it is merged to, and the value -1 is reserved for outlier clusters. In ODS, a unique increasing ID is assigned to each newly generated *normal-mc*, and a separate ID space is similarly used to track *outlier-mc* clusters. Each sample is then assigned the ID of the cluster it is merged to, and differently from wDBScan, ID is *not* reset across runs.

Plots in the top row of Fig. 7 depict the time evolution of the number of normal vs outlier clusters present in the window (wDBScan) or of the number of *normal-mc* vs *outlier-mc* (ODS). We observe that, after the initialization phase, both the models are composed by a single *normal-cluster* and no *outlier* ones. It is easy to observe that, in correspondence to the anomaly injection periods, the number of outlier clusters increase as expected.

The second plots row represents the ID of the clusters the samples are merged to. We observe that in both the models most of the samples are assigned a normal ID (generating a horizontal line). When an event is injected, algorithms flag samples as anomalous (red dots). In the case of ODS, outlier clusters may get promoted into normal clusters (which is visible, e.g., toward the end of the first event), whereas the old clusters are pruned over time due to the fading function.

The third and fourth rows illustrate the radius and the weight of the normal clusters respectively. One can easily observe from these plots the main differences in the model evolution of the two methods. On the one hand, after the injection of the first event, wDBScan generates a small cluster composed of 5 samples but, as soon as BGP convergence is achieved, returns merging new samples to the original cluster. The newly generated cluster, together with the anomalous samples, continue to be part of the model, even if not used for any purpose: i.e. no new samples are merged to the generated cluster as evidenced by the label ID (2nd plot), by the constant radius (3rd) and weight (4th) and also from the constant center (5th and 6th), which holds until they are excluded from the window. On the other hand, ODS removes old clusters (see the steady decrease of the weight of the cluster before pruning) and updates new ones (see the fast increase of the weight of the cluster after promotion) much more promptly than wDBScan.

The fifth and sixth rows show the first and second center components respectively, extracted through simple Principal Component Analysis (PCA) whose eigenvalues represents approximately 94% (wDBScan) and 77% (ODS) of the center. The two methods clearly behave differently: while ODS

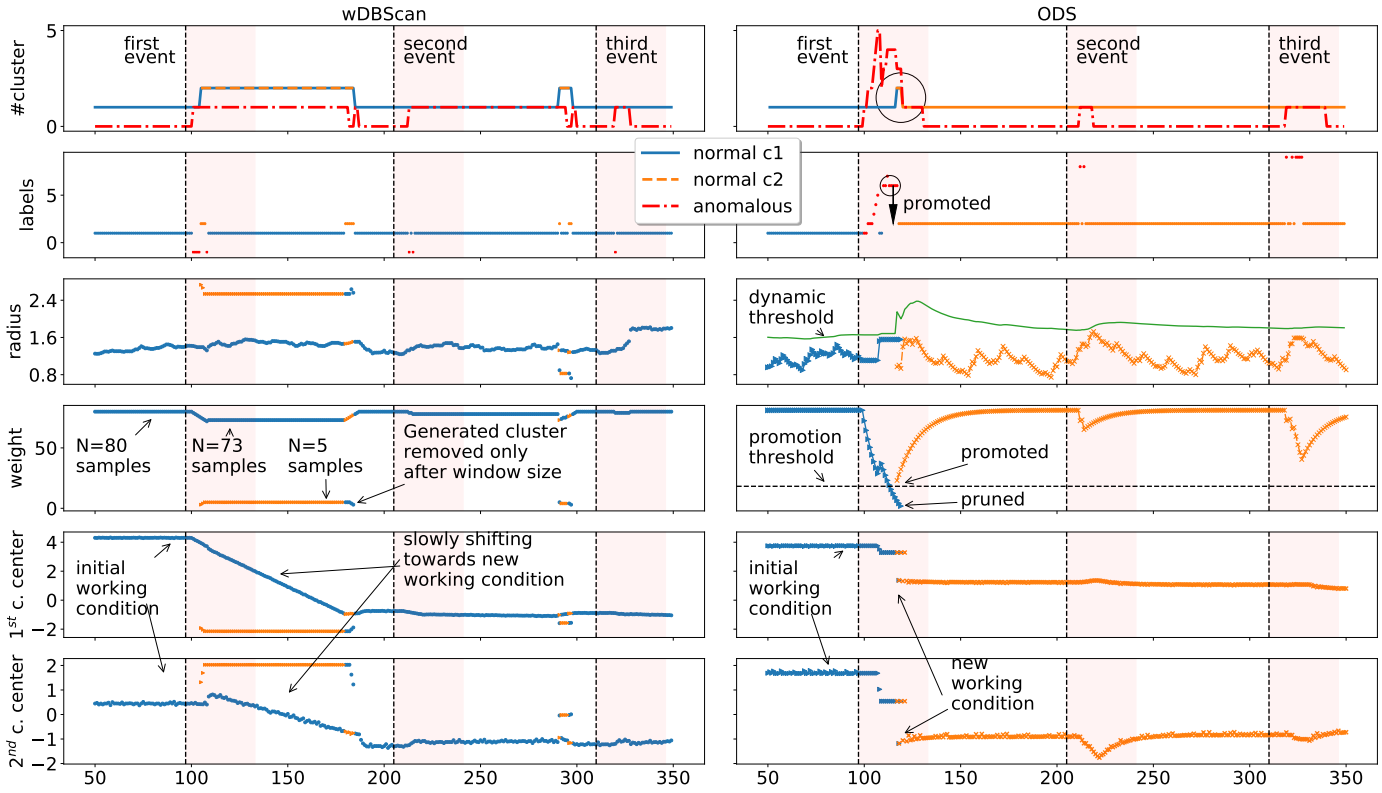


Fig. 7. wDBScan (left) vs ODS (right) model evolution over time. The top two rows discriminate *normal* vs *anomalous* clusters: the top row reports the number of *normal* vs *anomalous* clusters returned by the models, the second row reports the cluster ID to which each samples is merged to. The third and fourth rows show the radius and the weight evolution of each *normal* cluster respectively, while the two bottom ones depict the evolution of the first and second components extracted from the center of the clusters.

generates a new normal cluster (notice the pruning/promotion in the fourth row) in the region of the new working condition, wDBScan slowly drifts towards the latter. This effect depends on the parameters ϵ and w whose modification would however lead to a decrease in wDBScan performance (recall Fig. 4).

B. Detection Performance

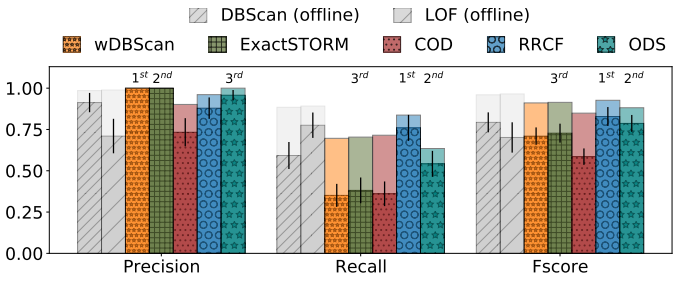
For validation, we rely on a set (E5, E9, E10) of experiments that are independent from those used for tuning (E2, E3), and are additionally well suited to stress test generalization capabilities of the studied methods. In particular, E5 is very similar to the tuning ones both in terms of traffic load (1 Tbps) as well as anomalies (BGP leaks) so it allows one to test the performances of the models in scenarios similar to those of the tuning phase. *E5 thus constitutes a good stress test of the generalization capabilities of the selected hyperparameter.*

E9 and E10 instead undergo different traffic loads (2.9 Tbps and 2 Tbps respectively) as well as different injected anomalies (Port Flaps caused by *interface shutdown from terminal* and *link failures caused by fiber pull* respectively). The latter ones allows thus one to test the performance of the models in scenarios very different from those used in the tuning phase. Additionally, while the events injected in E5 are severe ones and affect the operational status of an entire node (i.e. the node losses all the BGP tables and is no longer able to forward data on all the interfaces) and all the direct neighbors, the events in E9 and E10 affect a single interface which results in the loss of only one out of the four links between two nodes. We expect a mitigation of the effects produced in the network by the anomalies injected in E9 and E10 and for this reason we expect not all the nodes to detect the events. *E9 and E10 constitutes a good stress test of generalization capabilities of the algorithm.*

We compare algorithmic performance using several information retrieval metrics, notably Precision, Recall and $F_{0.5}$score in Fig. 8 and Accuracy, Markedness and Informedness in Fig. 9. We report the mean and confidence interval of each metric for the 7 algorithms tested, explicitly contrasting tuning (light opacity, background bars) and testing datasest (full opacity foreground bars). The information is additionally tabulated in detail in the figure, and an explicit annotation of the algorithm rank (limited to online algorithms) for each metrics is additionally reported to ease interpretation of the results.

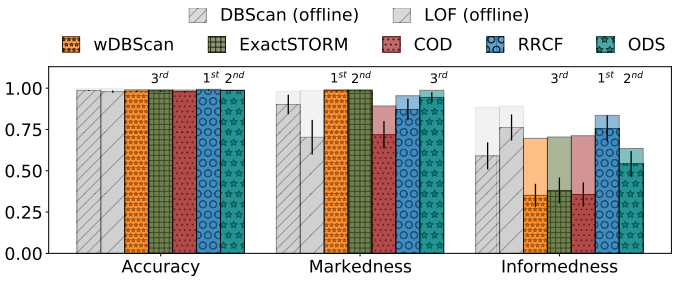
From a high-level viewpoint, considering the testing datasets we observe that RRCF stand out as being ranked 1st on four metrics (Recall, $F_{0.5}$, Accuracy and Informedness) – which makes it a very good choice. ODS instead stand out as the only algorithm being systematically ranked 2nd or 3rd on all 6 metrics – which makes it a robust choice.

In more details, we observe that the hyperparameters selected in the tuning phase yield to reasonably good performance in the test datasets as well, albeit performance diminish for all methods. This behavior is expected, as we know already



Dataset	Algorithm	Precision		Recall		$F_{0.5}$score	
Mean Tuning (E2/3)	*DBScan*	0.99±0.02		0.88±0.05		0.96±0.02	
	LOF	0.99±0.02		0.89±0.05		0.96±0.02	
	wDBScan	1.00±0.00	1st	0.70±0.07	3rd	0.91±0.03	2nd
	ExactSTORM	1.00±0.00	1st	0.70±0.07		0.91±0.02	2nd
	COD	0.90±0.06		0.72±0.07	2nd	0.85±0.06	
	RRCF	0.96±0.05		0.84±0.06	1st	0.93±0.04	1st
	ODS	1.00±0.00	1st	0.64±0.08		0.88±0.04	
Mean Test (E5/9/10)	*DBScan*	0.91±0.06		0.59±0.08		0.79±0.06	
	LOF	0.71±0.10		0.78±0.08		0.70±0.09	
	wDBScan	1.00±0.00	1st	0.35±0.07		0.71±0.08	
	ExactSTORM	1.00±0.00	1st	0.38±0.08	3rd	0.72±0.05	3rd
	COD	0.73±0.09		0.36±0.07		0.58±0.08	
	RRCF	0.88±0.06		0.76±0.08	1st	0.83±0.06	*1st*
	ODS	0.96±0.03	3rd	0.54±0.08	2nd	0.79±0.05	2nd

Fig. 8. Algorithms Performance Comparison: Precision, Recall and $F_{0.5}$ score. Figure and table report the average performance on the *testing* (full opacity, foreground bars) vs *tuning* (light opacity, background bars) dataset. The top-3 among the online algorithm are explicitly annotated.



Dataset	Algorithm	Accuracy		Markedness		Informedness	
Mean Tuning (E2/3)	*DBScan*	0.996±0.002		0.981±0.021		0.883±0.049	
	LOF	0.996±0.002		0.985±0.017		0.891±0.047	
	wDBScan	0.990±0.003	2nd	0.990±0.003	1st	0.696±0.069	3rd
	ExactSTORM	0.990±0.003		0.990±0.003	3rd	0.704±0.068	
	COD	0.988±0.003	3rd	0.892±0.064		0.712±0.065	2nd
	RRCF	0.993±0.003	1st	0.954±0.046		0.835±0.059	1st
	ODS	0.988±0.003	3rd	0.987±0.003	2nd	0.635±0.081	
Mean Test (E5/9/10)	*DBScan*	0.986±0.003		0.901±0.059		0.590±0.082	
	LOF	0.979±0.007		0.703±0.105		0.762±0.079	
	wDBScan	0.982±0.004		0.983±0.003	2nd	0.351±0.070	
	ExactSTORM	0.983±0.004	3rd	0.984±0.003	1st	0.381±0.078	3rd
	COD	0.978±0.004		0.718±0.083		0.356±0.074	
	RRCF	0.988±0.004	1st	0.871±0.065		0.756±0.079	1st
	ODS	0.986±0.003	2nd	0.944±0.031	3rd	0.541±0.078	2nd

Fig. 9. Algorithms Performance Comparison: Accuracy, Markedness and Informedness. Figure and table reports the average performance on the *testing* (full opacity, foreground bars) vs *tuning* (light opacity, background bars) dataset. The top-3 among the online algorithm are explicitly annotated.

E9 and E10 contain anomalies which are less disruptive and concern a single link, and are thus more difficult to detect. Discrepancy between tuning and testing phase is particularly visible considering the mean $F_{0.5}$ score over all datasets, where the degradation appears more severe for DBScan, wDBScan, LOF, ExactSTORM and COD. Investigating further, we find that DBScan and LOF performance degradation is due to low portability of parameter configuration as a function of the traffic load, which in turn has an impact on the distance between the samples. wDBScan, ExactSTORM and COD precision is consistent with that of the tuning phase, however, as a downside of the tradeoff, they achieve 0.35, 0.38 and 0.36 recall respectively.

In summary, RRCF and ODS appears to be a first and second choice respectively as for detection performance are concerned. Additionally, RRCF and ODS both appear to limit discrepancy with respect to the tuning phase, hinting to the fact that their parameters hardly fall into overfitting. Interestingly, RRCF and ODS stand at opposite sides in the recall (RRCF is top-1) vs precision tradeoff (RRCF is not even in the top-3), which makes both of them interesting options. For instance, in the case of ODS, precision could be traded for recall decreasing k_r, however this would increase the number of false alarms, which is not desirable in operational settings in our opinion, as otherwise alerts are unreliable and would thus simply be ignored. To further appreciate the differences between RRCF and ODS, we additionally contrast further metrics such as Precision@K, area under the received operating characteristics curve (AUC) and average precision (AP) in Tab.IV. From the comparison it emerges clearly that RRCF and ODS are practically indistinguishable for (at least) the top-5 events in each datasets, as shown by the Precision@3 and Precision@5 metrics, with RRCF exhibiting a better average precision over the whole set of anomalies.

TABLE IV
DETECTION PEFORMANCE: IN-DEPTH RRCF VS ODS COMPARISON

Metric	ODS	RRCF
Precision@3	0.899±0.063	0.898±0.021
Precision@5	0.776±0.088	0.786±0.024
Area Under the Curve (AUC)	0.967±0.016	0.989± 0.002
Average Precision (AP)	0.785±0.075	0.850± 0.017

C. Computational Complexity

With respect to SNMP polling in periods of 5 minutes, the 5 second sampling rate in the dataset considered in this work constitutes a 60× increase in the data velocity. In case future technological releases will reduce the sampling period further (to subsecond timescales), a major limiting factor will be then represented by the processing capabilities of the device and collectors. Under this angle, it is clear that computational complexity is of uttermost importance from a deployment perspective.

Yet, most of the well known methods used today in the literature are in large part unsuitable to process data fast enough both because they are computationally complex and have heavy resources requirements. In this section we experimentally measure the time complexity of *online* vs *offline* algorithms. We replicate E5 100× times, obtaining a stream composed by approximately 150K samples: by varying the length of the stream, we study the execution time trend of each method.

All the experiments are run on a server equipped with Linux Debian 10, Intel Xeon E5-1620 with 3.60 GHz CPUs and 32GB RAM. The scripts, available at [32], [56], use

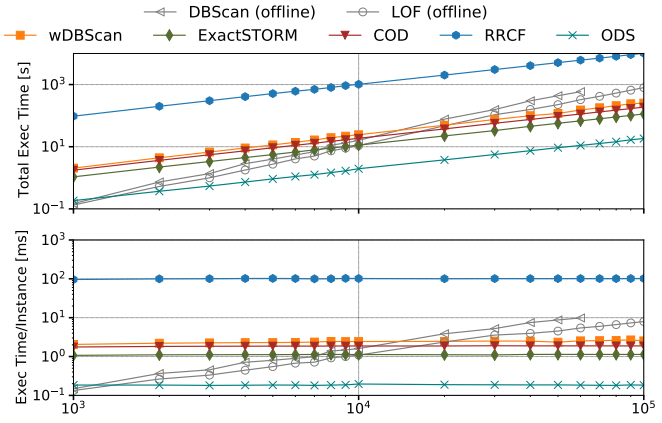


Fig. 10. Algorithm complexity: total execution time in seconds (top) and execution time per instance (bottom) for DBScan, LOF, wDBScan, ExactSTORM, COD, RRCF and ODS as a function of the dataset size. DBScan measurements are not complete as it runs out of memory for values greater than those shown in the plot.

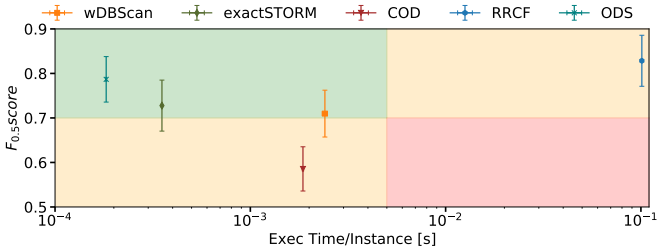


Fig. 11. Execution time per Instance (x-axis) vs Detection performance $F_{0.5}score$ (y-axis). The plot is annotated with semi-planes to better highlight the desirable corners of the design space: ODS sits at an interesting operational point for being significantly faster than all the algorithms tested and second only to RRCF in terms of information retrieval metrics ($F_{0.5}$ in this plot).

Python 3.8.3 [57], numpy 1.19.1 [58], pandas 1.05 [59] and scikit-learn 0.23.1 [60]. We use moreover RRCF's python implementation 0.4.3 present at [61].

Fig. 10 reports as a function of the stream length, the total execution time (top) and the average processing time of a single instance (bottom). ODS is the fastest one due to its linear complexity followed by ExactSTORM, COD, wDBScan, LOF and DBScan. DBScan is the one demanding most resources and reaches the memory limit for streams longer than 60K samples.

The bottom plot shows the processing time per instance. While DBScan and LOF show increasing processing time per instance per increasing stream size, online algorithms process the data in fixed amount of time per instance. Even by restricting our attention to online algorithms only, ODS stands out as being significantly faster than the others, followed by ExactSTORM (2×slower), COD (10×), wDBScan (13×) and RRCF (550×). By comparing the processing time and the sampling rate it is possible to establish an upper bound of the maximum sampling rate. Observing that the elapsed time per-sample is roughly 0.20 ms, our released ODS Python implementation is able to process approximately 5000 samples per second. Conversely, RRCF execution time requires about 100ms per sample, which caps its processing rate to at most 10 samples per second.

We summarize the complexity vs detection performance tradeoff in Fig.11 as a scatter plot of the execution time (in second, on the x-axis) vs the $F_{0.5}$ score performance (y-axis). Note that we use xyerrorbars, but the execution times confidence interval is very tight, and thus not visible due to the logarithmic x-axis scale. The plot is annotated with semi-planes (split so to halve the x-axis and y-axis ranges), to better highlight the desirable corners of the design space: top-left corner indicates algorithms that are both fast and good (green shading), whereas top-right corner indicates good but slow algorithms (yellow shading), bottom-left indicates fast but poorly performing algorithms (yellow shading) and finally bottom right indicates slot execution and poor performance (red shading). The picture clearly show that ODS sits at an interesting operational point for being significantly faster than all algorithms tested, and second (but nevertheless very close to) only to RRCF in terms of information retrieval metrics (as per Tab.IV).

VI. Discussion

We have contrasted a number of clustering methods for anomaly detection in networks. Whereas results testify stream-based approaches to be of interest, we aim at discussing here limitations and caveats to avoid pitfalls in their deployment.

Contextual anomalies. First, DenStream and consequently ODS are based on euclidean distances. We expect ODS to work on anomalies constituted by points *far* in the space, from the *normal* clusters and under no circumstances we do expect it to be able to detect *contextual* anomalies (e.g., such as absence of a periodic peak in a periodic signal). This suggests that techniques such as those studied here, should be complemented by others shall contextual anomalies be relevant in the deployment scenario.

Curse of dimensionality. The algorithm presented is of course not immune to the curse of dimensionality. This is likely to happen in practice whenever one would attempt to build a single model, aggregating several nodes and features per node. At the same time, model execution is extremely lightweight, which would allow to run multiple models in parallel, either at node-level (reducing communication complexity, but possibly missing events not detectable from internal measurements) or at feature-subset level (which would require some amount of communication between nodes, but possibly exploiting correlation among features at neighboring nodes).

Hyperparameters tuning. We have observed that hyperparameter tuning can lead to overfit, making deployment of unsupervised techniques difficult in practice, especially for methods whose parameters are closely related to the dataset (e.g. *contamination* in LOF or ϵ neighborhood in DBScan).

Instead, even though DenStream (and thus ODS) relies on four parameters, we have seen that it is possible to reduce their number by lumping some (e.g. by setting $\mu = \mu^+$) and by

dynamically setting others (e..g, so that the radius threshold $\epsilon = \bar{r} + k_r\sigma_r$ contains the bulk of the radius distribution).

At the same time, the fading factor λ (reduces gradually the importance of the samples) and the potential factor β (delimits the size of a *normal* cluster) must be selected with care. In particular, they are indeed bounded by $w > \beta/(1-2^{-\lambda})$ and therefore must be set taking into account their relationship and physical interpretation. For example, once λ is set, choosing a too high β could lead to the degenerate case in which the weight threshold is too high, and the model fails to build and maintain *normal-mcs*.

Autonomicity level ODS is not intended to completely replace a human network operator, but on the contrary it is a tool designed to facilitate his job. For instance, while ODS can update the model over time, it has to be initialized by a conscious operator, providing anomalous-free samples at at bootstrap. In turn, while ODS operates in the unmodified feature domain, there is a further need of explainable attention focus mechanisms [40] to let the human operator focus on the important features that triggered an event detection, an aspect orthogonal to our work.

VII. Conclusion

The recent emergence of model-driven telemetry opens new challenges for anomaly detection, and particularly makes the use of stream-based unsupervised machine learning tools very appealing. In this paper we develop, implement and open-source the ODS anomaly detection engine, based on the online clustering algorithm DenStream. We thoroughly analyze ODS on datasets gathered on BGP-only datacenter network loaded with up to 3 Tbps aggregated traffic, and extensively compare it with a set of offline (DBScan, LOF), windowed (wDBScan) and online techniques (ExactSTORM, COD and RRCF).

Our results show that despite ODS is apparently plagued with several parameters inherited from DenStream, their selection is quite straightforward, and performance are robust to inner parameter selection. Additionally, ODS is significantly faster than any of the tested algorithms, and second only to RRCF (yet very close to it) in terms of detection performance. Overall, the above results suggest ODS as a particularly lightweight and suitable algorithm for stream-mode real-time network anomaly detection.

Appendix

A. Available features

The available features are uniquely identified via their full YANG name, which in turn derives from the concatenation of an an *EncodingPath* with a *Leaf* among the available ones for that path (*Leafset*). Tab. V reports the full list of features used, that can be ascribed to either Data Plane (DP), for EncodingPaths matching `infra-statsd-oper` and `fib-common-oper` categories, or Control Plane (CP), for EncodingPaths matching the `ip-rib-ipv4-oper` and `ipv4-bgp-oper` categories. Extracting the full list of features per each interface and node yield about 750 data-plane and 25 control-plane features overall per dataset.

TABLE V
Available telemetry features

(CP) EncodingPath = `Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-names/ip-rib-route-table-name/protocol/bgp/as/information`
Leafset = {
active-routes-count
backup-routes-count
deleted-routes-count
paths-count
protocol-route-memory
routes-counts
}

(CP) EncodingPath = `Cisco-IOS-XR-ipv4-bgp-oper:bgp/instances/instance /instance-active/default-vrf/process-info`
Leafset = {
global__established-neighbors-count-total
global__neighbors-count-total
global__nexthop-count
global__restart-count
performance-statistics__global__configuration-items-processed
performance-statistics__global__ipv4rib-server__is-rib-connection-up
performance-statistics__global__ipv4rib-server__rib-connection-up-count
performance-statistics__vrf__inbound-update- messages
vrf__neighbors-count
vrf__network-count
vrf__path-count
vrf__update-messages-received
}

(DP) EncodingPath = `Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters`
Leafset = {
bytes-received
bytes-sent
}

B. Selected features

The full set of available features contains (i) redundant and totally correlated features (e.g. *free-memory* and *occupied-memory*), as well as (ii) categorical features design (e.g. *af-name*, *as* etc.) or (iii) features that are constant in our experiments (e.g., *output-queue-drops*). Out of the available features, we thus discard duplicated, categorical or constant features. Based on this process, we ultimately retain 64 DP and 18 CP features, for a total of 82 features. Notably DP features relate to the generic counters (i.e., whose EncodingPath match `interface/generic-counters`), whereas CP features relate to `bgp/as/information` and `vrf/process-info` EncodingPaths, which have proven useful for BGP anomaly detection in previous studies [28].

Acknowledgment

This work has been carried out at LINCS (http://www.lincs.fr), in the frame of a cooperation between Huawei Technologies France SASU and Telecom Paris. Any opinion, findings or recommendations expressed in this material are those of the author(s).

References

[1] M. Thottan and C. Ji, "Anomaly detection in ip networks," *IEEE Transactions on signal processing*, vol. 51, no. 8, pp. 2191–2204, 2003.
[2] Q. Wu, J. Strassner, A. Farrel, and L. Zhang, "Network telemetry and big data analysis," *IETF draft-wu-t2trg-network-telemetry-00*, Mar 2016.
[3] https://www.cisco.com/c/en/us/solutions/service-provider/cloud-scale-networking-solutions/model-driven-telemetry.html, 2018.
[4] https://www.arista.com/en/solutions/telemetry-analytics, 2018.
[5] http://support.huawei.com/enterprise/en/doc/EDOC1000173015?section=j006, 2018.
[6] M. Bjorklund, "YANG - A data modeling language for NETCONF," *RFC 6020*, Oct. 2010.
[7] "The yang 1.1 data modeling language," *RFC 7950* , Aug. 2016.
[8] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," *SIGCOMM Comput. Commun. Rev.*, vol. 34, Aug. 2004.
[9] Y. Huang, N. Feamster, A. Lakhina, and J. J. Xu, "Diagnosing network disruptions with network-wide analysis," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 1, pp. 61–72, Jun. 2007.

[10] S. Deshpande, M. Thottan, T. K. Ho, and B. Sikdar, "An online mechanism for bgp instability detection and analysis," *IEEE Transactions on Computers*, vol. 58, no. 11, pp. 1470–1484, Nov 2009.

[11] M. C. Ganiz, S. Kanitkar, M. C. Chuah, and W. M. Pottenger, "Detection of interdomain routing anomalies based on higher-order path analysis," in *ICDM*, 2006.

[12] D. S. Yeung, S. Jin, and X. Wang, "Covariance-matrix modeling and detecting various flooding attacks," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 37, no. 2, 2007.

[13] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang, "A novel anomaly detection scheme based on principal component classifier," in *IEEE Foundations and New Directions of Data Mining*, 2003.

[14] S. Rajasegarar, C. Leckie, and M. Palaniswami, "Hyperspherical cluster based distributed anomaly detection in wireless sensor networks," *Journal of Parallel and Distributed Computing*, vol. 74, no. 1, 2014.

[15] G. Münz, S. Li, and G. Carle, "Traffic anomaly detection using kmeans clustering," in *In GI/ITG Workshop MMBnet*, 2007.

[16] J. Mazel, P. Casas, R. Fontugne, K. Fukuda, and P. Owezarski, "Hunting attacks in the dark: clustering and correlation analysis for unsupervised anomaly detection," *International Journal of Network Management*, vol. 25, no. 5, pp. 283–305.

[17] P. Casas Hernandez, J. Mazel, and P. Owezarski, "Unsupervised Network Intruison Detection Systems: Detecting the Unknown without Knowledge," *Computer Communications*, vol. 35, no. 7, pp. 772–783, 2012.

[18] A. Lazarevic, L. Ertöz, V. Kumar, A. Ozgur, and J. Srivastava, "A comparative study of anomaly detection schemes in network intrusion detection," in *SIAM International Conference on Data Mining*, vol. 3, 05 2003.

[19] Z. Miller, B. Dickinson, W. Deitrick, W. Hu, and A. H. Wang, "Twitter spammer detection using data stream clustering," *Information Sciences*, vol. 260, pp. 64 – 73, 2014.

[20] J. Dromard, G. Roudière, and P. Owezarski, "Online and scalable unsupervised network anomaly detection method," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 34–47, 2017.

[21] Z. Miller, W. Deitrick, and W. Hu, "Anomalous network packet detection using data stream mining." *J. Information Security*, vol. 2, no. 4, 2011.

[22] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep 1995.

[23] H. Wang, J. Gu, and S. Wang, "An effective intrusion detection framework based on svm with feature augmentation," *Knowledge-Based Systems*, 09 2017.

[24] B. Subba, S. Biswas, and S. Karmakar, "A neural network based system for intrusion detection and attack classification," in *22nd National Conference on Communication (NCC)*, March 2016, pp. 1–6.

[25] G. Poojitha, K. N. Kumar, and P. J. Reddy, "Intrusion detection using artificial neural network," in *IEEE Conference on Computing, Communication and Networking Technologies*, July 2010.

[26] J. Li, D. Dou, Z. Wu, S. Kim, and V. Agarwal, "An internet routing forensics framework for discovering rules of abnormal BGP events," *ACM SIGCOMM Computer Communication Review*, vol. 35, pp. 55–66, 10 2005.

[27] I. O. de Urbina Cazenave, E. Köşlük, and M. C. Ganiz, "An anomaly detection framework for BGP," in *International Symposium on Innovations in Intelligent Systems and Applications*, June 2011.

[28] N. M. Al-Rousan and L. Trajković, "Machine learning models for classification of BGP anomalies," in *IEEE HPRS*, June 2012.

[29] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *SIAM Conference on Data Mining*, 2006.

[30] P. Lapukhov, A. Premji, J. Mitchell, "Use of BGP for Routing in Large-Scale Data Centers," in *RFC7938*, Aug. 2016.

[31] B. Al-Musawi, P. Branch, and G. Armitage, "Bgp anomaly detection techniques: A survey," *IEEE Communications Surveys Tutorials*, 2017.

[32] https://github.com/anrputina/ods-anomalydetection.

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

[34] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," *SIGMOD Rec.*, vol. 29, no. 2, May 2000.

[35] S. Guha, N. Mishra, G. Roy, and O. Schrijvers, "Robust random cut forest based anomaly detection on streams," in *ICML*, 2016, p. 2712–2721.

[36] F. Angiulli and F. Fassetti, "Detecting distance-based outliers in streams of data," in *ACM CIKM*, 2007, pp. 811–820.

[37] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos, "Continuous monitoring of distance-based outliers over data streams," in *IEEE ICDE*, 2011, pp. 135–146.

[38] D. Hawkins, *Identification of Outliers*. Chapman and Hall, 1980.

[39] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999.

[40] https://cloud.google.com/explainable-ai.

[41] J. M. Navarro and D. Rossi, "Hurra: Human-readable router anomaly detection," in *International Teletraffic Congress (ITC32)*, 2020.

[42] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *IEEE CISDA*.

[43] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[44] H. Wang and B. Raj, "On the origin of deep learning," 2017.

[45] Y.-W. Chen and C.-J. Lin, *Combining SVMs with Various Feature Selection Strategies*. Springer Berlin Heidelberg, 2006, pp. 315–324.

[46] Hanchuan Peng, Fuhui Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1226–1238, Aug 2005.

[47] A. Putina, D. Rossi, A. Bifet, S. Barth, D. Pletcher, C. Precup, and P. Nivaggioli, "Telemetry-based stream-learning of bgp anomalies," in *ACM SIGCOMM, Big-DAMA workshop*, 2018.

[48] J. MacQueen, "Some methods for classification and analysis of multivariate observations." University of California Press, 1967.

[49] C. Ning, C. An, and L.-X. Zhou, "An incremental grid density-based clustering algorithm," vol. 1313, pp. 1–7, 01 2002.

[50] A. Gunawan, "A faster algorithm for dbscan." in *Master's thesis, Technische University Eindhoven*, March 2013.

[51] 2018. [Online]. Available: https://github.com/cisco-ie/telemetry

[52] (2018) https://github.com/YangModels/yang.

[53] 2018. [Online]. Available: https://blogs.cisco.com/sp/introducing-pipeline-a-model-driven-telemetry-collection-service

[54] T. Green, A. Lambert, C. Pelsser, and D. Rossi, "Leveraging Inter-domain Stability for BGP Dynamics Analysis," in *International Conference on Passive and Active Network Measurement (PAM)*, 2018.

[55] P. Pébay, T. B. Terriberry, H. Kolla, and J. Bennett, "Numerically stable, scalable formulas for parallel and online computation of higher-order multivariate central moments with arbitrary weights," *Computational Statistics*, vol. 31, no. 4, pp. 1305–1325, Dec 2016.

[56] https://github.com/anrputina/ODS-2020.

[57] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[58] T. E. Oliphant, *A guide to NumPy*. Trelgol Publishing USA, 2006.

[59] W. McKinney *et al.*, "Data structures for statistical computing in python," in *9th Python in Science Conference*, vol. 445, 2010, pp. 51–56.

[60] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[61] M. Bartos, A. Mullapudi, and S. Troutman, "RRCF: Implementation of the Robust Random Cut Forest algorithm for anomaly detection on streams," *The Journal of Open Source Software*, vol. 4, no. 35, 2019.

Andrian Putina is a PhD candidate at Telecom Paris. He received his MSc degree in Information and Communications Technologies for Smart Societies from Politecnico di Torino, Italy in 2017. His current research interests are data mining, stream learning and anomaly detection.

Dario Rossi is Network AI CTO and Director of the DataCom Lab at Huawei Technologies, France. Before joining Huawei in 2018, he held Full Professor positions at Telecom Paris and Ecole Polytechnique and was holder of Cisco's Chair NewNet Paris. He has coauthored 10+ patents and 150+ papers in leading conferences and journals, that received 10 best paper awards, a Google Faculty Research Award (2015) and an IRTF Applied Network Research Prize (2016). He is a Senior Member of IEEE and ACM.