Telemetry-based stream-learning of BGP anomalies

Andrian Putina, Dario Rossi, Albert Bifet
Telecom ParisTech
name.surname@telecom-paristech.fr

Steven Barth, Drew Pletcher, Cristina Precup, Patrice Nivaggioli
Cisco Systems
name.surname@cisco.com

ABSTRACT

Recent technology evolution allows network equipments 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 at high-frequency. Processing this deluge of telemetry data in real-time clearly offers new opportunities for network control and troubleshooting, but also poses serious challenges.

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 BGP 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 by 1 Terabit/sec worth of real application traffic. In spirit with the recent trend toward reproducibility of research results, we make our code, datasets and demo available as open source to the scientific community.

CCS CONCEPTS

• **Information systems → Clustering**; **Data stream mining**; • **Networks → Network experimentation**; *Data center networks*;

ACM Reference Format:
Andrian Putina, Dario Rossi, Albert Bifet and Steven Barth, Drew Pletcher, Cristina Precup, Patrice Nivaggioli. 2018. Telemetry-based stream-learning of BGP anomalies. In *Big-DAMA'18: ACM SIGCOMM 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks , August 20, 2018, Budapest, Hungary*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3229607.3229611

1 INTRODUCTION

Nowadays network Operations and Management (OAM) increasingly rely 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*. 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 [23]. With SNMP, 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) [1–4, 24] has emerged as an interesting alternative to SNMP: instead 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, e.g., to trigger data publication when specific conditions are met.

Rather typically, a workflow common to several vendors (such as Cisco [1], Arista [2], Juniper [3] and Huawei [4]) is to express features via YANG [11, 12] 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 they are streamed – a challenging task at the heart of our work.

Anomaly detection is surely not a green field [14], however there are three key differences from this work and other related effort. Notably, we are first to leverage a dataset of features directly exported by network software via YANG, which makes the features more easily identifiable – and possibly portable across vendors.

Second, whereas there is an extensive literature on BGP anomaly detection (see [10] and references therein), the literature has mostly focused on the BGP inter-domain context. In this work we instead consider Content Service Provider (CSP) datacenters that, following recent trends[20], are designed with BGP as the only routing protocol. To the best of our knowledge, this study is the first to perform telemetry-based BGP anomaly detection on CSP networks.

Last but not least, we remark an important methodological difference: whereas there is a growing attention to data stream clustering techniques[16, 22], to the best of our knowledge DenStream has only seldom [18, 19] been used in network-related anomaly detection (raw headers are used as simple features in [18], whereas [19] deals with twitter spam). Summarizing our main contributions:

- (Sec.2) We engineer a realistic CSP testbed, loaded with up to 1 Tbps aggregate traffic, that we use to generate telemetry datasets annotated with manual ground truth about anomalous injected events. We make these datasets available at [5]
- (Sec.3) We devise an unsupervised anomaly detection algorithm based on DenStream [13] clustering that is apt at operating over a continuous stream of YANG telemetry data, offering its open-source implementation at [6];
- (Sec.4) We perform a thorough evaluation over the released datasets, gathering several findings related to both machine-learning (in terms of portability and accuracy) as well as aspects (such as feature selection and detection timeliness) relevant for network experts.

Big-DAMA'18, August 20, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5904-7/18/08…$15.00
https://doi.org/10.1145/3229607.3229611

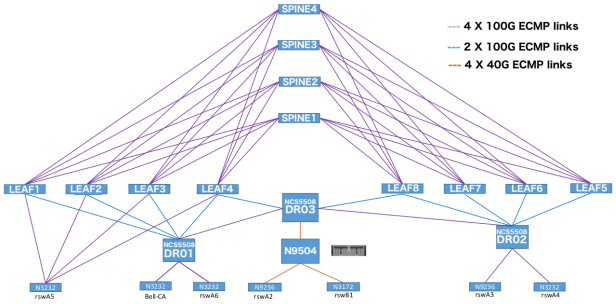

Figure 1: CSP Testbed topology

2 TESTBED AND DATASETS

Testbed. Our testbed replicates a traditional clos topology of a CSP datacenter. On the physical level, it is comprised of 8 leaf nodes interconnected via 4 spine nodes. 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 [20]. 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 1 Tbps of aggregated traffic. Though the testbed does not involve real users, it does use real equipment, protocols and applications typical of production networks. For instance, we configure the commercial traffic generators to use a mixture of TCP flows, G.711a IP voice call (64 kbps per call), RTP Blue Ray movie (45 Mbps per stream), Skype-1050P stream, AMR-WB VOIP (23.85 kbps per call), YouTube 4K video stream.

Data collection and labeling. We use the testbed to perform multiple experiments with different characteristics, that are listed in Table 1. At a sampling rate of 5 seconds, 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 gathered features are described via YANG models [7] and then extracted, decoded and stored by Pipeline [8] and made available as compressed CSV files. We point out that the dataset collection is growing, so that more datasets than those actually used in this paper are already available at [5].

The datasets have different *duration* and contain different *number* and *type* of events. We classify the working condition of our system 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 40 samples), after which we start injecting controlled anomalous events at randomized node locations. We inject anomalies spaced by either 120 or 300 seconds (depending on the dataset), and keep track of the injected event in a *ground truth* database, also available at [5].

In terms of BGP anomalies, we inject in the datasets the most common and frequent events that are found to impact BGP in a data center scenario: in particular, our datasets comprise BGP clear and port-flapping events. BGP clear basically resets the BGP process on a given device: operationally, this does occur at times when there are routing inconsistencies, or to "drain" a node prior to remove it from the forwarding plane for maintenance, or when there are issues associated with memory leaks that impact shared memory, which in turn impacts BGP as an artifact of memory loss. BGP clear can be a human driven or automated workflow, translating into the execution of a command (`bgp clear *`) on the target node.

Port-flapping (i.e., port state change) is something that happens in networks at a fairly high frequency for several reasons (including, e.g., logical causes such as administrative shutdown, as well as physical causes such as transceiver failure, laser fault, transceiver overheat, power loss, etc.). In our dataset, port-flapping is injected via either graceful administrative shutdown (software configuration, that can be precisely timed) or hardware anomaly (manually pulling the transceiver in the testbed, that remains precise given our 5 sec sampling period).

Table 1: Experimental datasets available at [5]

#	Traffic load	Anomalies no.	Type	Duration	Used for
0	0	0	–	1 h	Tuning ϵ (Sec. 4.1)
1	500Gbps	0	–	1 h	Tuning ϵ (Sec. 4.1)
2	1Tbps	11	BGP Clear	1 h	Tuning λ, β, μ (Sec. 4.1)
3	1 Tbps	8	BGP Clear	0.55 h	Tuning λ, β, μ (Sec. 4.1)
4	1 Tbps	5	Port Flap	0.72 h	Tuning λ, β, μ (Sec. 4.1)
5	1 Tbps	12	BGP Clear	2 h	Accuracy validation (Sec. 4.2) & feature selection (Sec. 4.3)
6	0	12	BGP Clear	2 h	Feature selection (Sec. 4.3)

Experiment description. While we control the anomaly injection process, there are two important points worth making. First, our methodology does not leverage ground truth information to build our data-driven models (i.e., as one would to in case of supervised classification), but we use this ground truth only to assess the precision and recall of our unsupervised methods (see Sec. 3-4).

Second, while we control the type of event, the node and time at which each anomaly is injected, we however do not control the *duration* of the anomalous event, which depends on the convergence time of the BGP protocol. Yet, the validation of an anomaly detection engine requires to define both the start-time (controlled and known) as well as the end-time (unknown) of the anomalous event: (i) shall the engine raise alarms during the anomalous event window, this yields to true positive detection, whereas (ii) if no alarm is raised during the window then this represent a false negative, and (iii) in case an alarm is raised outside the window, then it must be counted as a false positive. Expert knowledge suggests the BGP convergence process to take up to several minutes – usually less than 5 min, which we use in this work as "duration" of the anomalous event. In practice, as we shall see in Sec. 4, the detection delay is much faster and shorter than 0.5 min for all investigated settings – so that false alarms reported in this section are very likely not tied to our conservative choice of the anomalous window duration.

To assist reproducibility, Table 1 is also annotated with the specific section of this paper in which each particular dataset is used. Furthermore, detailed instructions on how to replicate our experiments is reported at [9] and additionally some of these datasets are readily accessible via the online demonstration [21] at https://telemetry.telecom-paristech.fr

Telemetry features. The features (in machine-learning terms) available in the testbed are a subset of the state available by YANG[12] models of the Cisco routers in the testbed. In a nutshell, YANG modules 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. All the leafs of the tree are features that could be exported in our testbed. At the same time, it is worth stressing that, e.g., routers in the testbed are equipped with Cisco IOS-XR 6.2.2, whose YANG hierarchy comprises over 378,000 lines, describing a hierarchy of over 45,000 features, with *nearly 5,000 types pertaining to the BGP protocol alone*. Since collecting and exporting features consumes CPU and bandwidth resources, it would be impossible to collect, for all nodes and interfaces, the totality of the supported features.

We thus perform a preliminary feature selection step, using domain-expertise to configure the most relevant features in the collection process, that are reported in the Appendix A for completeness. An example of data vs control plane features is depicted in Fig. 2 showing the time evolution of a control-plane (paths count, top) vs a data-plane (output load on an interface, bottom) feature during an experiment, depicting also the annotated ground truth (vertical red line for the start of anomaly, and shaded window for the anomaly duration). Interestingly, one can notice a strong positive (1st event), almost null (2nd) or slightly negative correlation (3rd) between these two features. Even though experimental results exhibit very good performance (Sec.4), confirming the experts choice as far as feature selection is concerned, we recognize the need for a more systematic selection process (discussed in Sec.5)

3 METHODOLOGY

Two families of approaches would fit our experimental datasets. The ground truth at our disposal would allow us to build detailed *supervised* multi-class models, precisely detecting the anomaly type; at the same time, these models would probably have a narrow application, as they would hardly be portable to different topologies, traffic matrices and load levels. Therefore, we prefer to adopt an *unsupervised* machine learning approach, to gather more coarse anomaly indications, that is less likely to incur the risk of overfitting the model to a particular dataset.

Under these premises, algorithms suitable for our problem include classic approaches such as K-Means [17] or DBScan [15] and stream-based approaches such as DenStream [13]. While K-Means is a very simple clustering algorithm based on centroids and distance between clusters, both DBScan and DenStream are density-based algorithms capable of discovering clusters of arbitrary shape. However, the original DBScan algorithm is designed to work on offline data, unlike DenStream where the model construction process is continuous. Additionally, DBScan has a computational complexity that is known to be $O(n \log(n))$ using optimized indexing data structures (but that can still fall back to $O(n^2)$ in the worst case), whereas DenStream has a $O(n + K)$ complexity (where K is a constant depending on its tuning). This is clearly appealing from an operational perspective (i.e., for applications that have to run indefinitely), for which we select DenStream (that we briefly describe in Sec. 3.1) as a basis for our anomaly detection system (Sec.3.2).

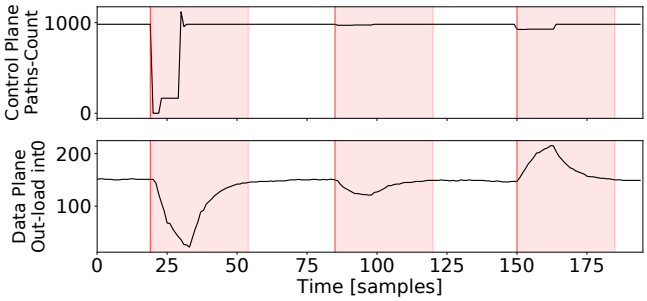

Figure 2: For the sake of illustration, and example of control plane (top) vs data plane (bottom) features, and annotated ground truth is depicted. The list of available YANG features is available in Appendix A.

3.1 DenStream

DenStream [13] is an algorithm designed for clustering in data streams: DenStream 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 $f(t) = 2^{-\lambda t}$, where $\lambda > 0$ 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_{ij} with creation time stamps T_{ij}. A MC is defined as a triple (w, c, r) where $w = \sum_j f(t - T_{ij})$ is the weight of the micro-cluster, $c = \frac{1}{w} \sum_j f(t - T_{ij}) p_{ij}$ is its center and $r = \frac{1}{w} \sum_j f(t - T_{ij}) d(c, p_{ij})$ its radius with $d(\cdot, \cdot)$ the Euclidean distance. By breaking clusters into MCs, DenStream allows one 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 merges it to the nearest *core* MC provided that the radius of the merged cluster does not exceed a given threshold ϵ; otherwise, DenStream attempts to merge the point to the closest *outlier* MC, and a new outlier MC is finally created if the merge fails. In our context, when a sample is merged to a *core* MC, we consider the sample as *normal* (and *anomalous* otherwise).

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 – which makes it suitable for dynamic environments. Although DenStream introduces a number of parameters (namely, λ, ϵ, β and μ) we show in Sec. 4.1 that their tuning is relatively straightforward and that DenStream performance are quite robust for a large range of settings.

3.2 Telemetry-based anomaly detector

We define two simple criteria for raising alarms based on DenStream clustering, which we experimentally evaluate in Sec. 4.2 using the classic key performance indicators of information retrieval.

Temporal order κ_T. : *each node operates independently and an alarm is raised only upon reception of* κ_T *consecutive outlier samples at a node.* In particular, a true positive detection (false alarm) occurs when κ_T consecutive samples are labeled as *anomalous* and the ground truth labels the system as being in anomalous (normal) state. The parameter κ_T trades off precision for recall and delay: intuitively, increasing κ_T reduces the amount of false alarms, but at the same time reduces the overall amount of raised alarms, and mechanically inflates the delay by a factor of κ_T.

Spatial order κ_S. : *all nodes operate altogether and an alarm is raised when, during the same time slot, at least* κ_S *nodes label a sample as an outlier.* As before, the ground truth labels assist in evaluating the accuracy of the method. Intuitively, increasing κ_S trades off high precision for lower recall (as anomalies that affect fewer than κ_S nodes in the same time-slot can go unnoticed), although the detection delay is minimized in this case.

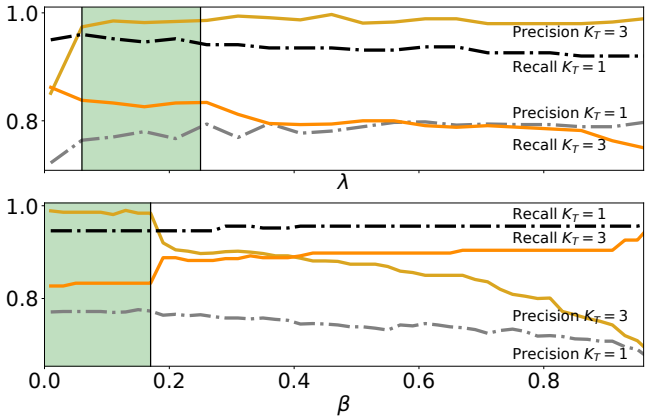

Figure 3: Sensitivity analysis of the fading λ and potential β factors (datasets #2-#4).

4 EXPERIMENTAL RESULTS

4.1 DenStream sensitivity

We start by tuning the parameters of the DenStream algorithm, using datasets #0–#4 described in Tab. 1.

Radius threshold ϵ. DenStream performance depends on the choice of the radius parameter ϵ. A machine learning expert would advocate for a *dynamic* selection of the threshold, that is automatically computed as the radius of the cluster obtained clustering together the first $S = 40$ samples at the beginning of the model construction. We contrast this choice with an *fixed* threshold setting: a network expert may suggest to compute it as the radius of the cluster obtained during an experiment in which no anomalies are injected (datasets #0–#1) and tested on the remaining datasets (#2–#5). Experiments (not shown for lack of space) show that the use of a fixed threshold reduces recall and precision by over a factor of 2×. The radius depends on the normal working condition of the network such as traffic load, number of neighbors, the topology, etc., which makes a fixed selection fragile and impractical. Dynamic threshold selection is easy to compute (as few samples suffices) and relevant as it is, by construction, portable to any scenario.

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, and the rule for outlier MC promotion becomes $w > \beta/(1 - 2^{-\lambda})$.

Fading λ **and Potential** β **factors**. The only free parameters in our setup are λ and β, which both have a physical interpretation. 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 spatial interpretation, as *it determines the minimum number of samples needed for outlier MC promotion to core MC.*

We point out that a machine-learning expert would select λ and β as a result of a "grid selection" procedure, whereas a network domain expert could be tempted to select λ and β according to physical properties of the system. 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 core MC ($\beta = 0.17$) 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 $\lambda \approx 0.111$. We perform a sensitivity analysis and question these choices: Fig. 3 shows the precision and recall results varying $\lambda \in [0, 1]$ with fixed $\beta = 0.05$ (top) as well as for varying $\beta \in [0, 1]$ for fixed $\lambda = 0.15$ (bottom). Without loss of generality, we show results for two values of the temporal order $\kappa_T \in \{1, 3\}$.

Several important takeaways can be gathered from the graphs. First, the performance metrics (Precision and Recall) are smoothly varying over λ and β. The green shaded area depicts a region where the performance is stable, suggesting a window of values were λ and β can be chosen. Second, the performance metrics are more sensitive to the detection order K_T than by λ and β. This is a very desirable result since, for an operator using the system described in Sec.4.2, the detection order is a more intuitive knob to tune than the multiple parameters of the underlying DenStream algorithm. Third, as far as the BGP timescale is concerned, the domain experts choice for λ falls in the acceptable range that a machine-learning expert would suggest. Fourth, it is preferable to let DenStream produce small micro-clusters (low β), as this allows the model to more closely follow the system dynamics.

A last remark is worth making. While we have seen that performance metrics are smoothly varying on β and λ, we point out that their selection is still primarily correlated with the telemetry sampling rate: as such, in case of device reconfiguration and especially for very different sampling timescales, a new sensitivity analysis would be highly recommended (if not mandatory).

κ_T	Precision	Recall	False Alarm	F-measure	Delay [s]
$\kappa_T = 1$	0.78	1	0.35	0.88	9.9
$\kappa_T = 2$	0.89	1	0.17	0.94	17.6
$\kappa_T = 3$	1	0.90	0	0.95	24.6
$\kappa_T = 4$	1	0.84	0	0.91	30.1
$\kappa_T = 5$	1	0.78	0	0.87	35.5

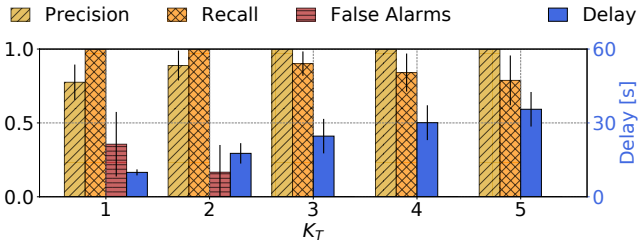

Figure 4: Temporal detection KPI for varying κ_T (dataset #5)

κ_S	Precision	Recall	False Alarm	F-measure	Delay [s]
$\kappa_S = 1$	0.50	1	1	0.66	5.6
$\kappa_S = 2$	0.92	1	0.17	0.96	7.3
$\kappa_S = 3$	1	1	0	1	9.0
$\kappa_S = 4$	1	1	0	1	10.2
$\kappa_S = 5$	1	1	0	1	10.6

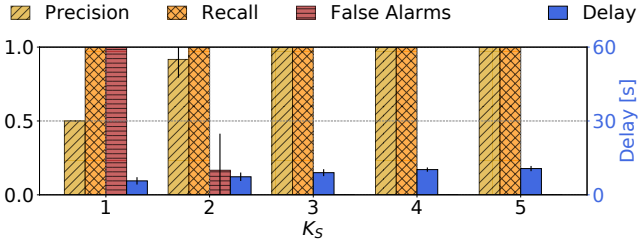

Figure 5: Spatial detection KPI for varying κ_S (dataset #5)

4.2 Temporal κ_T vs spatial κ_S orders

We now assess the portability of our settings: we use the parameters selected according to datasets #0-#4, and validate the selection by assessing the system accuracy on a different dataset #5. In particular, Fig. 4 and Fig. 5 illustrate the KPIs obtained for temporal κ_T and spatial κ_S detection criteria respectively (additionally tabulating results for the sake of readability). Precision, recall and false alarms exhibit a similar trend for both κ_T and κ_S. As expected, requiring multiple consecutive outliers from the same node ($\kappa_T > 1$) induces a sizeable delay, since $\kappa_T \geq 3$ samples are needed to avoid false alarms. Additionally, recall slightly decreases for increasing κ_T as the criterion becomes excessively restrictive for our datasets. Conversely, $\kappa_S \geq 3$ allows perfect recall and precision with only very limited additional delay (smaller than 10 sec). While we do not observe false negatives in our dataset, it is possible for anomalies local to a single node to go unnoticed – in which case κ_T may be more appropriate. Therefore, depending on the operator preferences on the above metrics, a reasonable parameter choice falls into $\kappa_{(S|T)} \in \{2, 3\}$.

	Precision			Recall			False Alarms		
	CP	CP+DP	DP	CP	CP+DP	DP	CP	CP+DP	DP
$K_T = 2$	1	0.89	0.87	1	1	1	0	0.17	0.19
$K_T = 3$	1	1	1	0.80	0.90	0.72	0	0	0
$K_S = 2$	1	0.92	0.88	1	1	1	0	0.17	0.25
$K_S = 3$	1	1	1	1	1	1	0	0	0

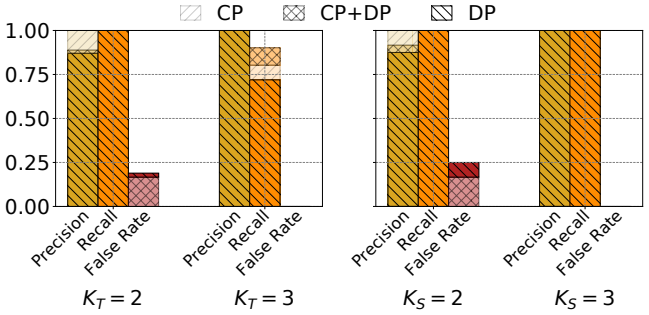

Figure 6: Domain expert feature selection: Data Plane (DP) only vs Control Plane (CP) only vs all DP+CP features (dataset #5)

4.3 Feature selection

Finally, we perform, on the same dataset, a "domain expert" feature selection by using control plane (CP) only, data plane (DP) only or all DP+CP features. Fig. 6 shows the KPIs considering $\kappa_{(S|T)} \in \{2, 3\}$. An interesting observation can be learned from the experiment: using only CP-related information, *no false alarms are raised already for $\kappa_S = 2$ and $\kappa_T = 2$*. Conversely, using only DP-related features generally yields to poor performance, while the use of DP+CP leads to intermediate performance even though DP features also bring valuable information beyond the noise (notice that CP+DP increases recall for $\kappa_T = 3$). In case of CP-only features, we repeat the experiment using a dataset with no synthetic user traffic (dataset #6), obtaining accuracy results that are indistinguishable from practical purposes – the largest absolute discrepancy across all KPIs over all $K_{(S|T)}$ values is smaller than 4%.

These experiments suggest the following. First (CP-only case), the algorithm is reliable in detecting anomalies *even with very few features* (25 out of nearly 5,000) related to the protocol under investigation, testifying its precision. Second (DP-only case), the very same algorithm is still able to detect BGP anomalies *even without having any information on the BGP protocol itself*, testifying its robustness.

5 DISCUSSION

Anomaly detection is certainly not a green field, but, the recent emergence of model-driven telemetry opens new challenges, and makes the use of stream-based unsupervised machine learning tools very appealing. In this paper: (i) we engineer a full blown testbed representative of a BGP-only datacenter network of a Content Service Provider, that we load with up to 1 Tbps aggregated traffic; (ii) we use the testbed to produce datasets with annotated ground truth, collecting hundreds of features from tens of devices, that we make available to the scientific community[5]; (iii) we develop, implement, open-source and analyze an anomaly detection engine

based on the DenStream [13] clustering algorithm, further making available detailed instructions to reproduce our experiments at[9].

Our results show that: (i) despite DenStream is apparently plagued with several parameters, their selection is quite straightforward, and performance are robust to inner parameter selection; (ii) low spatial and temporal orders are sufficient to jointly attain high recall, high precision and low delay; (iii) detection of BGP-related anomalies is more effectively based on control-plane only features, which is due to the implicit separation of timescale between data plane and control plane.

These results opens up future work, along the following directions: (i) given the low computational complexity of DenStream, it would be advisable to run several models in parallel, each of which would detect anomalies from different protocols/layers/planes; (ii) individual learners could be complemented with a macroscopic model, that would learn from alarms of individual learners, as opposed to the raw telemetry features; (iii) from a practical point of view, a systematic experimental study of the maximum number of features that can be exported is necessary.

ACKNOWLEDGEMENTS

We thank our shepherd Felix Iglesias and the anonymous reviewers whose useful comments helped us ameliorating this paper. This work was carried out at LINCS and was funded by NewNet@Paris, Cisco's Chair "Networks for the Future" at Telecom ParisTech (https://newnet.telecom-paristech.fr).

REFERENCES

[1] 2018. (2018). https://www.cisco.com/c/en/us/solutions/service-provider/cloud-scale-networking-solutions/model-driven-telemetry.html
[2] 2018. (2018). https://www.arista.com/en/solutions/telemetry-analytics
[3] 2018. (2018). https://www.juniper.net/documentation/en_US/junos/topics/concept/junos-telemetry-interface-oveview.html
[4] 2018. (2018). http://support.huawei.com/enterprise/en/doc/EDOC1000173015?section=j006
[5] 2018. (2018). https://github.com/cisco-ie/telemetry
[6] 2018. https://github.com/anrputina/OutlierDenStream. (2018).
[7] 2018. https://github.com/YangModels/yang. (2018).
[8] 2018. (2018). https://blogs.cisco.com/sp/introducing-pipeline-a-model-driven-telemetry-collection-service
[9] 2018. (2018). https://github.com/anrputina/OutlierDenStream-BigDama18
[10] B. Al-Musawi, P. Branch, and G. Armitage. 2017. BGP Anomaly Detection Techniques: A Survey. *IEEE Communications Surveys Tutorials* 19, 1 (2017), 377–396.
[11] M. Bjorklund. 2010. YANG - A data modeling language for NETCONF. *RFC 6020* (Oct. 2010).
[12] M. Bjorklund. 2016. The YANG 1.1 Data Modeling Language. *RFC 7950* (Aug. 2016).
[13] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. 2006. Density-based clustering over an evolving data stream with noise. In *2006 SIAM Conference on Data Mining*.
[14] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM Comput. Surv.* 41, 3 (2009), 15.
[15] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *ACM KDD*.
[16] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han. 2014. Outlier Detection for Temporal Data: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 26, 9 (Sept 2014), 2250–2267.
[17] J. MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proc. Berkeley Symposium on Mathematical Statistics and Probability*.
[18] Zachary Miller, William Deitrick, and Wei Hu. 2011. Anomalous Network Packet Detection Using Data Stream Mining. *J. Information Security* 2, 4 (2011), 158–168.
[19] Zachary Miller, Brian Dickinson, William Deitrick, Wei Hu, and Alex Hai Wang. 2014. Twitter spammer detection using data stream clustering. *Information Sciences* 260 (2014), 64 – 73.
[20] P. Lapukhov, A. Premji, J. Mitchell. 2016. Use of BGP for Routing in Large-Scale Data Centers. (Aug. 2016).
[21] A. Putina and D. Rossi et al. 2018. Unsupervised real-time detection of BGP anomalies leveraging high-rate and fine-grained telemetry data. In *IEEE INFOCOM, Demo Session*.
[22] Jonathan A. Silva, Elaine R. Faria, Rodrigo C. Barros, Eduardo R. Hruschka, André Carvalho, and João Gama. 2013. Data Stream Clustering: A Survey. *ACM Comput. Surv.* 46, 1 (July 2013), 13:1–13:31.
[23] Marina Thottan and Chuanyi Ji. 2003. Anomaly detection in IP networks. *IEEE Transactions on signal processing* 51, 8 (2003), 2191–2204.
[24] Q. Wu, J. Strassner, A. Farrel, and L. Zhang. 2016. Network Telemetry and Big Data Analysis. *IETF draft-wu-t2trg-network-telemetry-00* (Mar 2016).

A YANG MODEL FEATURES

Exporting the full hierarchy of over 45,000 YANG features available with Cisco IOS-XR 6.2.2 has a clearly prohibitive cost, for which we resort to an expert-driven selection of a small subset of features. Full YANG names of each feature are derived by concatenating an *encoding path* with a *leaf* among the available ones for that path. Roughly, the computational complexity of telemetry features grows with the number of encoding paths to be accessed; however, the cost of accessing one, several (or even all) leafs under the same path is then very similar: thus, we decide to focus on the selection of encoding paths, exporting then all leafs under that path.

The set of exported features used in this paper is manually configured based on expert knowledge, selecting (i) 25 leaf features our of the nearly 5,000 available for the BGP protocol alone, and (ii) 25 data plane features for each of the 32 interfaces, plus (iii) a set of 20 per-node data plane features. Notice that only numeric features can be used in DenStream (e.g., all control-plane features ending with `-name` are categorical). For reason of space we limitedly report in this appendix the full list of Control Plane (CP) features available in our dataset, and invite the reader to [5] for more details and the list of data plane features.

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
- af-name
- as
- backup-routes-count
- deleted-routes-count
- paths-count
- protocol-route-memory
- route-table-name
- routes-counts
- saf-name
- vrf-name

}

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
- instance-name
- 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-name
- vrf__neighbors-count
- vrf__network-count
- vrf__path-count
- vrf__update-messages-received

}