基于群集的负载平衡实现更好的网络安全

Cluster-Based Load Balancing for Better Network Security

Gal Frishman
IDC

Yaniv Ben-Itzhak
IBM Research

Oded Margalit
IBM Research

ABSTRACT

In the big-data era, the amount of traffic is rapidly increasing. Therefore, scaling methods are commonly used. For instance, an appliance composed of several instances (*scaled-out* method), and a load-balancer that distributes incoming traffic among them. While the most common way of load balancing is based on round robin, some approaches optimize the load across instances according to the appliance-specific functionality. For instance, load-balancing for scaled-out proxy-server that increases the cache hit ratio.

In this paper, we present a novel load-balancing approach for machine-learning based security appliances. Our proposed load-balancer uses clustering method while keeping balanced load across all of the network security appliance's instances. We demonstrate that our approach is scalable and improves the machine-learning performance of the instances, as compared to traditional load-balancers.

CCS CONCEPTS

• **Information systems** → **Clustering**; • **Security and privacy** → **Intrusion detection systems**; • **Networks** → **Network security**; *Middle boxes / network appliances*; • **Computing methodologies** → **Machine learning**;

KEYWORDS

Network Security, Machine Learning, Misuse Detection, Flow Distribution, Load-Balancing, Clustering

ACM Reference format:
Gal Frishman, Yaniv Ben-Itzhak, and Oded Margalit. 2017. Cluster-Based Load Balancing for Better Network Security. In *Proceedings of Big-DAMA '17, Los Angeles, CA, USA, August 21, 2017,* 6 pages.
https://doi.org/10.1145/3098593.3098595

1 INTRODUCTION

Load-balancing techniques are prevalent in enterprise and data-center networks, and mostly used to support scaled-up services. Traditionally, load-balancer (LB) is used for flows distribution between web-servers, where the ingress flows are required to be distributed across a cluster of servers. The load-balancing is usually employed by either round-robin (e.g., Round-Robin DNS [13]), IP level (e.g, [3]), or according to the current servers' load [2]. Other load-balancers (LBs) are employed over layer 4 and layer 5 (e.g., [6, 22]). LBs are also used for other network services, such as network proxy servers. Such LBs are usually based on the proxy-server's cache-content and their goal is to increase the cache hit ratio rather than achieve a balanced load (e.g. [32]).

Machine learning (ML) based network security appliances (e.g., network intrusion detection system – NIDS) are inherently different from traditional network services, such as web and proxy servers. Web servers respond to queries; and proxy servers cache data, while providing high cache-hit rate. On the other hand, network security appliances generate statistics, maintain different phases (training/non-training), and generate prediction or classification based on their collected statistics during the training phase. Therefore, optimizing the performance of a network security appliance that consists of several instances requires different load-balancing considerations.

Machine learning misuse detection based NIDS, which searches for known intrusive patterns, have been extensively researched and presented. For instance: Naive Bayes[21], Hidden Naive Bayes[16], Support Vector Machines (SVM) [10, 18], K-Nearest Neighbors [1], Decision Trees [8, 15], Random Forest [33], Multilayer Perceptron (MLP) [20], back-propagation neural network [5], Self Organization Maps (SOM) [14], and evolving fuzzy neural networks [4].

Most of the current work on misuse detection based NIDS present a centralized solution, in which a single central device is used to analyze and detect patterns for all of the ingress network traffic. However, in the big-data era, it would be infeasible or extremely expensive for a single device to train, process and cluster/predict all of the data-centers traffic. In order to cope with the increased ingress traffic, other works proposed distributed ML algorithms (e.g., [11, 29]), which share information between them. Another approach is using an NIDS appliance, which is composed of several 'centralized ML-based' instances, where each instance independently analyzes subset of the network traffic. Hence, no shared information (states or sync) is required between the instances. Practically, this approach can utilize the previous centralized misuse NIDS solutions.

In this paper, we present a cluster-based load-balancer for the latter approach, which aim to maximize misuse detection performance of NIDS, while preserving a relatively balanced load among its instances. Our load-balancer assigns a group of similar flows to the same instance, by using ML clustering algorithm. 'Similar flows' are defined by having some correlation between them, according to given features (e.g., same source subnet).

2 MOTIVATION

Nowadays, networks are required to support high capacity demands and to scale as needed. Furthermore, the SDN and NFV paradigms are shifting the industry from using vendor-specific physical network appliances to virtual appliances deployed over standard COTS servers. Therefore, all network functions, including NIDS, are being virtualized and scaled.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Big-DAMA '17, August 21, 2017, Los Angeles, CA, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5054-9/17/08…$15.00
https://doi.org/10.1145/3098593.3098595

There are several known ways to scale services [9]:

Scale up. By using stronger server with more memory and compute resources to support the growing bandwidth demands. However, this approach may not be feasible since: a) Some machine learning algorithms can't be parallelized and the compute capacity of a single core is limited. b) The training time may grow faster than linear with the processed network demand.

Scale out by distributed algorithm over several instances of the same NIDS appliance [11, 29]. This approach is more scalable by design; however, it results in additional network overhead in order to synchronize the appliance's instances.

Scale out by load-balancing the network traffic over multiple independent NIDS instances. The load balancer should distribute traffic across the instances in a way that *maximizes overall learning performance* (i.e., corresponds to detection quality), while maintaining a balanced load as possible.

In this paper, we focus on the latter approach. We argue that conventional load-balancing approaches (e.g., round-robin [7] and uniform random flow distributions), which aim for equal load over the instances, are not suitable for machine-learning based NIDS. Such conventional approaches degrade the overall machine learning performance (in terms of the prevalent ML metrics: accuracy, precision, recall, F-score, and area under curve). The performance of ML NIDS appliance highly depends on the similarity across different network flows in the traffic assigned to each of its instances. Therefore, optimizing the security performance of such NIDS appliance requires new approach for load-balancing the network traffic between its instances. Our work presents a new approach to distribute groups of similar network flows across the appliance's instances (with the same functionally) to achieve better security performance, while preserving a relatively balanced load among its instances.

Finding and generalizing such a load-balancing approach is not a trivial task. First, research domain is relatively broad. There are many clustering algorithms that can be used by the load balancer and many misuse detection algorithms that can be used by the NIDS. Proper evaluation requires good understanding of existing methods and their respective domains (e.g., statistical, centralized/distributed, offline/online, classification based, outlier based, knowledge based, soft computing based methods). In addition, many of the proposed methods have no publicly available implementation for evaluation purposes.

Second, availability and quality of public datasets in this domain is limited. KDD CUP '99 dataset [27] is outdated and heavily criticized for not representing attacks realistically. NSL-KDD dataset [28] is more challenging for ML but still lacks modern attacks. Therefore, in order to develop load-balancing approach that is dataset agnostic, variety of traces and data sources should be used.

Third, the combinations of the different required evaluation steps is growing exponentially. There are several clustering methods to be employed as the LB; different feature extraction/selection/preprocessing methods for preparation the input data to the ML algorithms; and several misuse algorithms to be employed as the NIDS. Therefore, we eliminate unfeasible combinations by using different filtering criteria for each of the evaluation steps.

3 EVALUATION

In this section, we evaluate our approach; i.e., the improvement gained by using ML-based clustering load-balancer for misuse ML-based NIDS. First, we describe our evaluation setup (i.e., the traces, models, tools and method). Then, we present ML performance comparison to both centralized baseline model and distributed model by round-robin based LB. Lastly, we demonstrate the scalability of our approach.

Traces: For evaluation, the well known NSL-KDD [28] dataset is used. It is reduced version of KDD CUP '99 [27], more adequate for machine learning benchmarks. Each record represents a sequence of packets in a TCP session. We use the original NSL-Train+/Test+ (20%) sets for training/testing.

The features include statistics on all connections to same destination during last 2 seconds and previous 100 connections, statistics of the current TCP session, and expert knowledge. Records are labeled as either *Normal* or one of attack classes: *DOS* (denial of service), *PROBE* (scanning), *R2L* (remote-to-local, unauthorized remote access), or *U2R* (user-to-root, unauthorized local access).

Features Preprocessing: Since the evaluated misuse detection algorithms require numerical features, categorical features are either dropped or encoded using one-hot encoding. The numerical features are either left as is, or scaled according to min-max values.

We evaluate two different models:

Baseline Model (Centralized): All traffic is processed by a single NIDS instance . The ML performance serves as a baseline for comparison with the *scaled-out* model. The performance metrics, in terms of accuracy, precision, recall, F-score and area under curve (AUC) are produced using 5-fold cross validation over test data.

Scaled-out Model: The traffic is load-balanced across multiple instances of same NIDS appliance, by different load-balancing approaches. First, we evaluate round-robin and unified random approaches that achieve balanced load across all instances, which ignore the functionality of the ML instances – *conventional LB*.

Second, we compare the conventional LB approaches to our *cluster-based LB* approach, which clusters incoming traffic, and assigns each cluster to NIDS instance. To that end, we define a *clustering model* which consists of clustering algorithm (table 1), clustering features set (table 2), and specific features preprocessing settings – i.e., *clustering model=clustering-algorithm+features-set+features-preprocessing*.

The *scaled-out* model is evaluated for load-balancing, ML performance metrics and scalability, as depicted in the sequel.

Tools and Method: The evaluation framework is based on scikit package[1], running on Python 2.7, Amazon Linux 64-bit, Intel Xeon E5-2666 v3 @ 2.9GHz, 7.5GB RAM.

The evaluation process is depicted in figure 1(a). First, the dataset is split to train and test sets. Then, the model is trained for every *clustering model*. The load-balancer clustering model is evaluated for varying number of clusters k={3..9}.

The evaluation process of a *clustering model* for specific k is depicted in figure 1(b). Each cluster is assigned to NIDS instance, and all instances operate in parallel with the same misuse detection algorithm. Each instance is trained and tested using 5-fold cross

[1] a free software machine learning library for the Python programming language [23] (http://scikit-learn.org).

Approach	Algorithms
Conventional Load Balancer	Round-Robin, Unified-Random
Cluster-Based Load Balancer	K-means
Scaled-out Misuse Detection NIDS Algorithms	1-NN, Decision-Tree, Random-Forest, Multi-Layer Perceptron, SVM (only for min-max scaled features).

Table 1: The evaluated LB, clustering and misuse detection algorithms for the baseline and scaled-out models.

validation. The confusion matrices and prediction probabilities of all instances are summarized into a single matrix, which is used for the calculation of the ML performance measurements: accuracy, precision, recall, F-score and area under curve (AUC).

Clustering Models Feasibility: In order to define whether a clustering model is feasible, we first define a load-balanced clustering model by maximum standard deviation ($\sigma = 6000$) for cluster sizes and maximum ratio of 25 between the biggest and the smallest clusters, over all number of clusters (i.e., over k={3..9}). Only load-balanced clustering models are evaluated, and unbalanced clustering models are omitted.

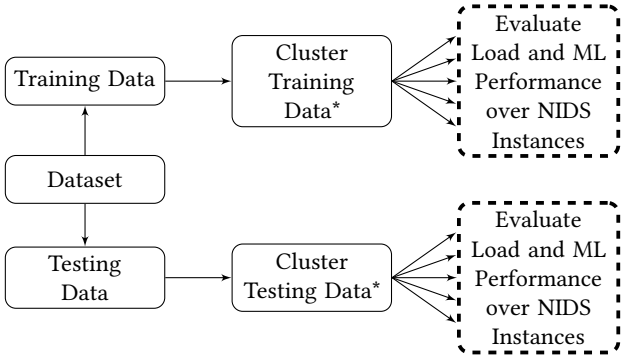
Second, we check the feasibility of the ML algorithms employed by the NIDS instances for each clustering model. For instance, SVM requires that every cluster must have samples of more than one class. Therefore, any clustering model which does not satisfy this requirement is not evaluated (e.g., K-means clustering model for features options 1, 2).

Under this criteria, the feasible clustering models are feature sets 3 and 7 (table 2) with all preprocessing and encoding options, and feature set 5 with min-max scaling and one-hot encoding. Figure 2 presents the maximal ratio between the biggest and the smallest clusters for every feasible clustering model. Further load-balancing improvements of the cluster-based LB are left for future-work (see §5 for more details).

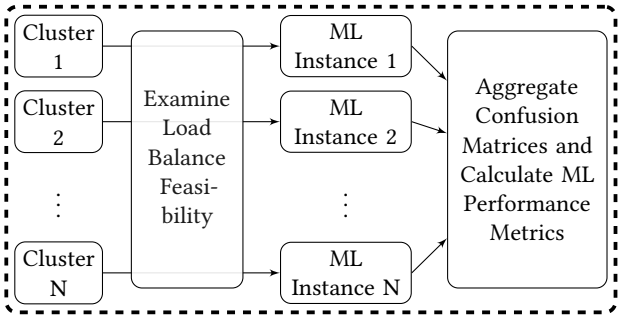
ML Performance Comparison: Our cluster-based LB significantly improves some of the ML-based misuse detection algorithms (table 1), while the improvement for other algorithms is less significant. We mainly use F-score metric as it considers both precision and recall scores and it is nearly identical to the accuracy scores in our evaluation.

Figure 3 presents the average F-score for decision tree algorithm. It can be seen that the baseline and conventional LB approaches achieve average F-score of approx. 0.8 while our approach achieves significant improvement: all feasible clustering options achieve F-score of above 0.9. The ML performance improvement of random forest algorithm is very similar.

On the other hand, figure 4 presents comparison of the average F-score for SVM ML algorithm. It can be seen that SVM gain little to no improvement, as compared to the baseline and conventional LB approaches (the same also applies for 1-NN algorithm). Furthermore, multi-layer perceptron performance gain is somewhere in the middle with an average F-score of approx 0.87 for conventional LB, 0.91 for baseline and 0.92-0.96 for cluster-based LB.



(a) Workflow of the scaled-out model evaluation (* - using the *same* clustering model)



(b) Workflow of the load and ML performance evaluation over the NIDS instances

Figure 1: The evaluation workflow

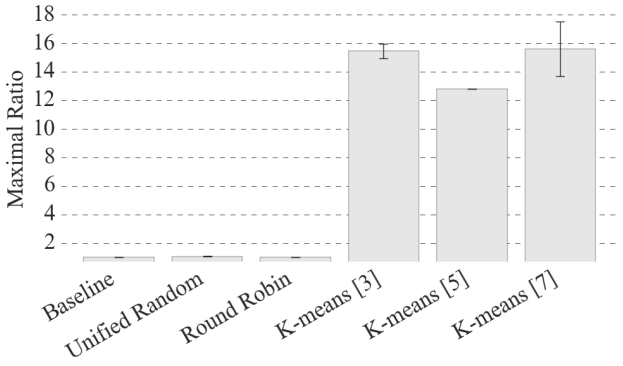


Figure 2: Maximal ratio between the biggest and the smallest cluster sizes, over all cluster sizes (k). Comparison between baseline, conventional LB, and clustering LB (our approach).

The previous figures present an averaged results. Therefore, in the following, we zoom-in and present some of our evaluation measurements for specific settings and metrics, in order to further demonstrate and emphasize our approach improvements. Figure 5 presents the average area under curve of the decision-tree ML algorithm, for the U2R class, the rarest class of attack and therefore the most challenging to learn. The cluster-based LB improves the

Clustering Feature Sets	# of Features	Description
1. Same destination	5	Statistics on connections to same destination in last 2 seconds
2. Same service	4	Statistics on connections to same service in last 2 seconds
3. 100 connections same destination	10	Statistics on 100 last connection to same destination
4. Domain expert	13	Features within a connection suggested by domain knowledge
5. TCP features	9	Statistics on individual TCP connection
6. All 2 secs features	9	Union of 2 secs same destination + service
7. All history features	19	Union of 100 connections + all 2 secs features

Table 2: Features options for clustering. Each row describes a set of features for traffic clustering.

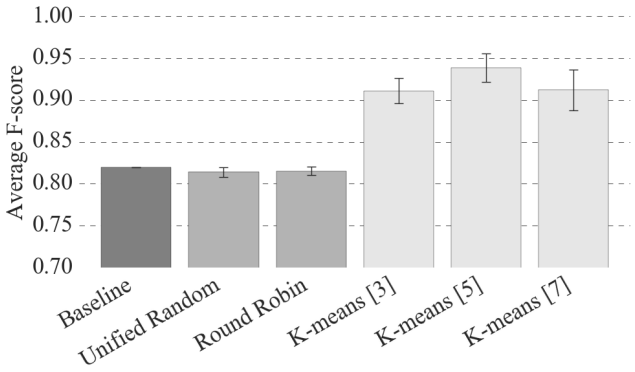


Figure 3: Decision tree average F-score. Comparison between baseline, conventional LB, and clustering LB – our approach.

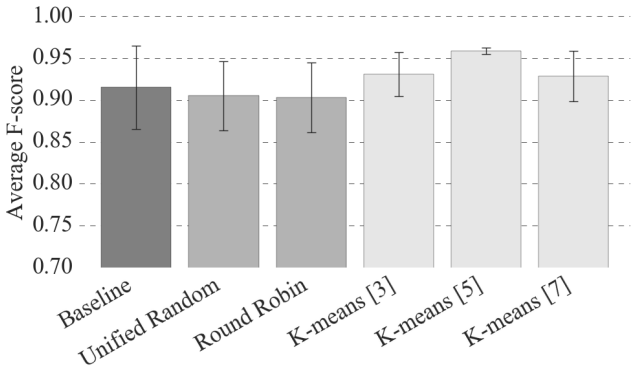


Figure 4: SVM average F-score comparison (same results obtained for 1-NN).

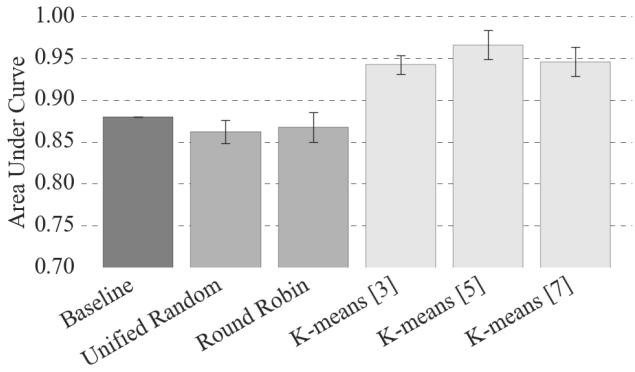


Figure 5: Area under curve (AUC) of the decision-tree ML algorithm for U2R class.

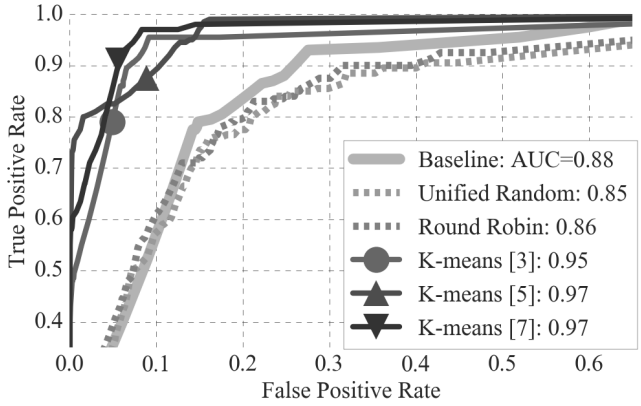


Figure 6: ROC curves and areas of Decision-Tree ML algorithm for U2R (user-to-root) class with 9 instances (k=9). Numbers in the legend indicates the AUC of each clustering model.

AUC by approx. 10% as compared to the baseline and conventional LB approaches. Figure 6 presents the ROC curves and AUC of decision-tree ML algorithm for k=9. It can clearly be seen that all ROC curves are improved by the cluster-based LB, as observed by the F-score and AUC measurements in figures 3 and 5, respectively.
Discussion: One might think that since all of baseline model's flows are centrally processed, it's performance should be the highest (either in terms of F-score or AUC). However, our evaluation demonstrates otherwise, and in particular for Decision-Tree based NIDS (see figures 3 and 5). The Decision-Tree algorithm is configured with three levels. We assume that the cluster-based LB actually employs the upper decision level. Hence, each NIDS instance employs decision-tree algorithm after the first decision level is already processed. Therefore, effectively our cluster-based LB

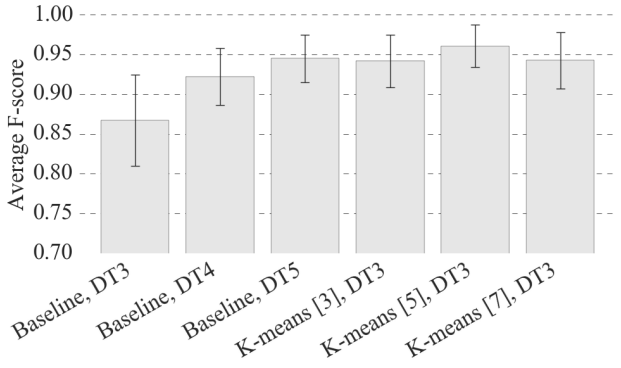
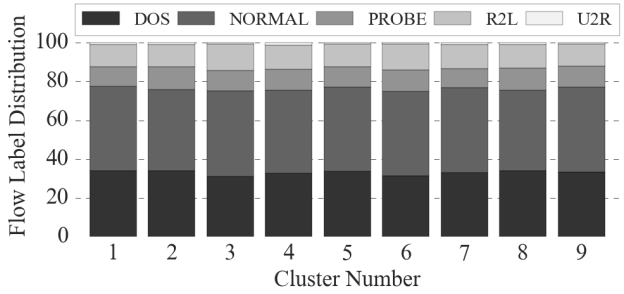
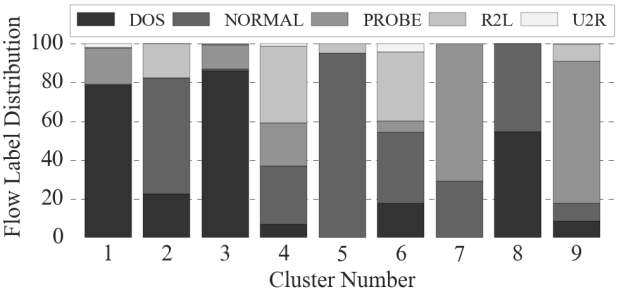


Figure 7: Comparison of the achieved average F-score of decision-tree between between the baseline model – centralized processing with three to five levels (DT3, DT4, DT5), and cluster-based LB with only three levels.



(a) Flow label distribution for round-robin.



(b) Flow label distribution for K-means [3].

Figure 8: Comparison of the flow label distribution over the clusters between conventional round-robin LB and cluster-based LB, for k=9.

achieves performance of decision-tree with higher number of levels, even though each instance is configured with three levels. Figure 7 demonstrates our assumption of the effective improvement of the number of levels for decision-tree by the cluster-based LB. It presents the average F-score of the baseline centralized approach for decision-tree with three to five levels (DT3, DT4, and DT5, respectively). The achieved average F-score of decision-tree with three levels (DT3) when using the cluster-based LB is equal to the achieved F-score of decision-tree with five levels (DT5) when centrally processed according to the baseline model. Therefore, employing the cluster-based LB effectively achieves F-score results of centralized decision tree with five levels, while practically using decision tree with only three levels.

K	3	4	5	6	7	8	9
F-score	0.922	0.929	0.932	0.934	0.936	0.938	0.940
Std. Dev	0.044	0.037	0.034	0.032	0.031	0.029	0.029

Table 3: K-means scalability

Figure 8 presents the flow labels distribution for round-robin LB and cluster-based LB with k=9. It can be seen that the flow labels distribution of round-robin over the clusters is quite uniform. Whereas, the K-means based LB results in clusters, in which each instance contains higher percentage of a specific flow label; hence, allow for better training for this specific flow label. Thus, in overall achieves better ML-performance (e.g., as presented in figure 6).

Scalability: Our evaluation shows that all of the feasible clustering options scale well with k (number of clusters). Table 3 demonstrates the scalability of K-means with numerical features scaled according to min-max values. For every k, the F-score is averaged over all feasible clustering models and misuse detection algorithms. It can be seen that the average F-score slightly improves as number of clusters increases.

4 RELATED WORK

Previous works presented clustering and flow correlation methods for NIDS. However, none of which were used for scale-out by LB as presented in this paper. Lin et al. in [19] present clustering method which is used to improve supervised classification performance and run times. However, this method is not suitable for the load balancing problem as number of instances is derived from number of attack classes, and not from traffic capacity needs.

Several works [12, 17, 24, 30, 31] propose different load balancing methods and improvements for NIDS, which is not based on ML classifiers. For instance, Le et al. [17] aims to maximize the correlation between flows of same node while keeping equal load across nodes. However, their NIDS model is based on statistical algorithm (CUSUM) for detecting DDoS attacks.

5 SUMMARY AND FUTURE WORK

In this paper, we present our first step towards integrating ML-based load-balancer for NIDS. Our proposed LB approach uses clustering method in order to improve the machine-learning performance of NIDS appliances. We demonstrated that our approach is scalable and achieves better ML performance as compared to conventional load-balancers.

This work can be extended in several directions, which include:

Load-Balancing Clustering Improvements. Our cluster-based LB approach should be improved in terms of performance by obtaining better load-balanced clusters, which can be achieved by several ways: a) *Optimization techniques:* In this paper the features sets for clustering in table 2 were selected manually. However, any

other features combination, which can be achieved for instance by optimization methods, may further improve the misuse detection trade-off between balanced load and the ML performance of the security instances. b) *Feedback from the NIDS instances* to the LB regarding load and traffic similarity can be used on-line in order to improve the clusters both in terms of load-balancing and clusters accuracy. c) *Other clustering algorithms:* such as fuzzy C-means[2] can be used in order to improve the ML-based NIDS performance.

On-Line Evaluation. In this work, we presented an off-line evaluation for ML clustering based load-balancer for NIDS. However, an on-line clustering and misuse detection algorithms are required in order to handle ingress traffic in real systems. To that end, algorithms such as on-line K-means can be used.

Anomaly-Detection NIDS. Misuse detection algorithms require training on labeled traffic, which may be hard to obtain. Pure anomaly detection algorithms such as one class SVM, iForest and EXPoSE [25] require only benign traffic and may be more suitable for real life applications.

Additional Traces. In this paper we conduct our evaluation on NSL-KDD dataset. In our future work, we will use other traces, such as ISCX 2012 dataset [26], which is more modern and contains actual traffic traces.

REFERENCES

[1] Adebayo O Adetunmbi, Samuel O Falaki, Olumide S Adewale, and Boniface K Alese. 2008. Network intrusion detection based on rough set and k-nearest neighbour. *International Journal of Computing and ICT Research* 2, 1 (2008), 60–66.

[2] Luis Aversa and Azer Bestavros. 2000. Load balancing a cluster of web servers: using distributed packet rewriting. In *Performance, Computing, and Communications Conference, 2000. IPCCC'00. Conference Proceeding of the IEEE International*. IEEE, 24–29.

[3] Azer Bestavros, Mark Crovella, Jun Liu, and David Martin. 1998. Distributed packet rewriting and its application to scalable server architectures. In *Network Protocols, Proceedings. Sixth International Conference on*. IEEE, 290–297.

[4] Sampada Chavan, Khusbu Shah, Neha Dave, Sanghamitra Mukherjee, Ajith Abraham, and Sugata Sanyal. 2004. Adaptive neuro-fuzzy intrusion detection systems. In *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*, Vol. 1. IEEE, 70–74.

[5] Shreya Dubey and Jigyasu Dubey. 2015. KBB: A hybrid method for intrusion detection. In *Computer, Communication and Control (IC4), 2015 International Conference on*. IEEE, 1–6.

[6] Daniel E Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. 2016. Maglev: A fast and reliable software network load balancer. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, 523–535.

[7] F5. 2015. Deploying the BIG-IP LTM with Multiple BIG-IP AAM and ASM Devices. (2015). https://www.f5.com/pdf/deployment-guides/ltm-asm-aam-dg.pdf

[8] Shekhar R Gaddam, Vir V Phoha, and Kiran S Balagani. 2007. K-Means+ ID3: A novel method for supervised anomaly detection by cascading K-Means clustering and ID3 decision tree learning methods. *IEEE Transactions on Knowledge and Data Engineering* 19, 3 (2007).

[9] Guilherme Galante and Luis Carlos E de Bona. 2012. A survey on cloud computing elasticity. In *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*. IEEE, 263–270.

[10] Shi-Jinn Horng, Ming-Yang Su, Yuan-Hsin Chen, Tzong-Wann Kao, Rong-Jian Chen, Jui-Lin Lai, and Citra Dwi Perkasa. 2011. A novel intrusion detection system based on hierarchical clustering and support vector machines. *Expert systems with Applications* 38, 1 (2011), 306–313.

[11] Weiming Hu, Jun Gao, Yanguo Wang, Ou Wu, and Stephen Maybank. 2014. Online adaboost-based parameterized methods for dynamic distributed network intrusion detection. *IEEE Transactions on Cybernetics* 44, 1 (2014), 66–82.

[12] Wenbao Jiang, Hua Song, and Yiqi Dai. 2005. Real-time intrusion detection for high-speed networks. *Computers & security* 24, 4 (2005), 287–294.

[13] Eric Dean Katz, Michelle Butler, and Robert McGrath. 1994. A scalable HTTP server: The NCSA prototype. *Computer Networks and ISDN systems* 27, 2 (1994), 155–164.

[14] H Gunes Kayacik, A Nur Zincir-Heywood, and Malcolm I Heywood. 2003. On the capability of an SOM based intrusion detection system. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, Vol. 3. IEEE, 1808–1813.

[15] Gisung Kim, Seungmin Lee, and Sehun Kim. 2014. A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. *Expert Systems with Applications* 41, 4 (2014), 1690–1700.

[16] Levent Koc, Thomas A Mazzuchi, and Shahram Sarkani. 2012. A network intrusion detection system based on a Hidden Naïve Bayes multiclass classifier. *Expert Systems with Applications* 39, 18 (2012), 13492–13500.

[17] Anh Le, Ehab Al-Shaer, and Raouf Boutaba. 2008. On optimizing load balancing of intrusion detection and prevention systems. In *INFOCOM Workshops 2008, IEEE*. IEEE, 1–6.

[18] Yinhui Li, Jingbo Xia, Silan Zhang, Jiakai Yan, Xiaochuan Ai, and Kuobin Dai. 2012. An efficient intrusion detection system based on support vector machines and gradually feature removal method. *Expert Systems with Applications* 39, 1 (2012), 424–430.

[19] Wei-Chao Lin, Shih-Wen Ke, and Chih-Fong Tsai. 2015. CANN: An intrusion detection system based on combining cluster centers and nearest neighbors. *Knowledge-based systems* 78 (2015), 13–21.

[20] Dima Novikov, Roman V Yampolskiy, and Leon Reznik. 2006. Anomaly detection based intrusion detection. In *Information Technology: New Generations, 2006. ITNG 2006. Third International Conference on*. IEEE, 420–425.

[21] Mrutyunjaya Panda and Manas Ranjan Patra. 2007. Network intrusion detection using naive bayes. *International journal of computer science and network security* 7, 12 (2007), 258–263.

[22] Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg, David A Maltz, Randy Kern, Hemant Kumar, Marios Zikos, Hongyu Wu, et al. 2013. Ananta: cloud scale load balancing. In *ACM SIGCOMM Computer Communication Review*, Vol. 43. ACM, 207–218.

[23] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12, Oct (2011), 2825–2830.

[24] Lambert Schaelicke, Kyle Wheeler, and Curt Freeland. 2005. SPANIDS: a scalable network intrusion detection loadbalancer. In *Proceedings of the 2nd Conference on Computing Frontiers*. ACM, 315–322.

[25] Markus Schneider, Wolfgang Ertel, and Fabio Ramos. 2016. Expected similarity estimation for large-scale batch and streaming anomaly detection. *Machine Learning* 105, 3 (2016), 305–333.

[26] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. 2012. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security* 31, 3 (2012), 357–374.

[27] Salvatore J Stolfo, Wei Fan, Wenke Lee, Andreas Prodromidis, and Philip K Chan. 2000. Cost-based modeling for fraud and intrusion detection: Results from the JAM project. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, Vol. 2. IEEE, 130–144.

[28] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. 2009. A detailed analysis of the KDD CUP 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*. IEEE, 1–6.

[29] Daxin Tian, Yanheng Liu, and Yang Xiang. 2009. Large-scale network intrusion detection based on distributed learning algorithm. *International Journal of Information Security* 8, 1 (2009), 25–35.

[30] Matthias Vallentin, Robin Sommer, Jason Lee, Craig Leres, Vern Paxson, and Brian Tierney. 2007. The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 107–126.

[31] Konstantinos Xinidis, Ioannis Charitakis, Spyros Antonatos, Kostas G Anagnostakis, and Evangelos P Markatos. 2006. An active splitter architecture for intrusion detection and prevention. *IEEE Transactions on Dependable and Secure Computing* 3, 1 (2006), 31–44.

[32] Philip Shi-lung Yu. 2002. Loading balancing across servers in a computer network. (Feb. 26 2002). US Patent 6,351,775.

[33] Jiong Zhang, Mohammad Zulkernine, and Anwar Haque. 2008. Random-forests-based network intrusion detection systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38, 5 (2008), 649–659.

[2] a form of clustering in which each data point can belong to more than one cluster (also referred to as *soft clustering*).