The Interdisciplinary Center, Herzlia

Efi Arazi School of Computer Science

M.Sc. program - Research Track

Cluster-Based Load Balancing for Better Computer Network Security

by

Gal Frishman

M.Sc. dissertation, submitted in partial fulfillment of the requirements for the M.Sc. degree, research track, School of Computer Science
The Interdisciplinary Center, Herzliya

January 2018

This work was carried out under the supervision of Dr. Yaniv Ben-Itzhak from IBM Research Israel, with guidance of Prof. Anat Bremler-Barr from the Efi Arazi School of Computer Science, The Interdisciplinary Center, Herzliya.

Some results in this thesis have been published as articles by the author and research collaborators in conferences and journals during the course of the author's masters research period, the most up-to-date versions of which being:

Gal Frishman et al. Cluster-based load balancing for better network security. In *Proceedings of Big-DAMA '17*. ACM, 2017.

Acknowledgements

I would like to thank my supervisor, Dr. Yaniv Ben-Itzhak, for the guidance, encouragement and advise he provided throughout my time as his student. Thanks to him, I had an amazing experience presenting the work in *Big-DAMA '17*.

I would also like to thank Dr. Oded Margalit for reviewing the paper and providing insights on the research.

Lastly, I must express my gratitude to my wife, Ariella, for her continued support and encouragement, for pushing me to finish the research and thesis writing and providing me the time and peace of mind to do that.

Abstract

In the big-data era, the amount of traffic is rapidly increasing. Therefore, scaling methods are commonly used. For example, 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 example, load-balancing for scaled-out proxy-server that increases the cache hit ratio.

In this paper, we present a novel load-balancing approaches for machine-learning based security appliances. Our proposed load-balancers use clustering methods while keeping a 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.

The contribution of this work is threefold. First, we present an up to date survey of balance-driven clustering algorithms and discuss their applicability to the load balancing problem. Second, we evaluate and demonstrate the intrusion detection performance improvement of clustering based load-balancers over traditional load-balancers. Third, we open source our evaluation framework and implementation of different clustering algorithms for reproducibility and future use.

Contents

List of Figures

Abbreviations and Notations

1NN	:	1 Nearest Neighbors
AUC	:	Area Under Curve
CLB	:	Clustering Load Balancer
COTS	:	Commercial Off-The-Shelf
DT	:	Decision Tree
LB	:	Load Balancer
MCBC	:	Modified K-means Clustering algorithm with Balancing Constraints
ML	:	Machine Learning
MLP	:	Multilayer Perceptron
MMR	:	Min-Max Ratio
NFV	:	Network Function Virtualization
NIDS	:	Network Intrusion Detection System
RF	:	Random Forest
ROC	:	Receiver Operating Characteristic
RR-DNS	:	Round Robin Domain Name System
SDN	:	Software Defined Network
SVM	:	Support Vector Machine

Chapter 1

Introduction

1.1 Background

Load-balancing techniques are prevalent in enterprise and data-center networks, and mostly used to support scaled-up services.

Early works [KBM94, Mog95] are based on Round-Robin DNS (RR-DNS) to distribute incoming connections across a cluster of servers. Other well-known load-balancing approaches are based on IP level according to flow tuple [BCLM98], or according to the relative load on the different network instances [AB00]. Other load-balancers are employed on Layer 4 and Layer 7; for instance, HAProxy [Tar06]. 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 between the servers (e.g., [Yu02]).

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, using the following classification algorithm: Naïve Bayes[PP07], Hidden Naïve Bayes[KMS12], Support Vector Machine (SVM) [HSC$^+$11, LXZ$^+$12], K-Nearest Neighbors [AFAA08], Decision Tree [KLK14], Random Forest [ZZH08].

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-center's traffic. In order to cope with the increased ingress traffic, other works proposed distributed ML algorithms (e.g., [HGW$^+$14, TLX09]), which share information between NIDS' instances. Another approach is to use NIDS appliance 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 five different clustering based load-balancers for the latter approach, which aim to maximize misuse detection performance of NIDS, while preserving a balanced load among its instances.

Our load-balancers cluster incoming traffic into groups of similar flows and assign them to NIDS' instances, one group per instance. 'Similar flows' are defined by having some correlation between them, according to given features (e.g., same source subnet).

1.2 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.

There are several known ways to scale services [GdB12]:

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. For example, [HGW+14, TLX09] train distributed nodes locally and use them to train a global model. This approach is more scalable by design; however, it requires 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 [F515] 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: precision, recall, F-score, and area under curve). Previous proposals for NIDS load balancing [LASB08, VSL+07, XCA+06, SWF05, JSD05] are inapplicable too as they are designed for non-learning NIDS (e.g., signature based or using statistical properties).

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 new approaches to distribute groups of similar network flows across the appliance's instances (with the same functionally) to achieve better security performance, while preserving a 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). 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 [SFL$^+$00] is outdated and heavily criticized for not representing attacks realistically. NSL-KDD dataset [TBLG09] 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.

1.3 Problem Statement

The role of load balancers is to maintain a balanced load across instances of some application, typically by controlling how incoming traffic is distributed. In this work, we focus on ML-based NIDS as the application. On every NIDS instance an intrusion detection classifier is trained independently of the other instances. The load balancer for this application has two goals:

1) Distribute network flows in a way that maximizes NIDS' learning performance (e.g., its precision, recall, F-score, etc.).

2) Maintain a balanced load across the instances.

To this end, we define a *clustering load balancer* (CLB). To address the first goal, we assume and demonstrate that network flows of same normal/attack class have intrinsic similarities. The CLB tries to unearth these similarities using a clustering algorithm. It clusters incoming traffic into groups of similar flows and assigns them to NIDS' instances, one group per instance. Therefore, traffic of same class is expected to end up on a single or very few instances, compared with traffic distributed using traditional load balancing methods (e.g., uniform random or round robin). Thus, ML classifiers trained on these groups can capture more of the underlying characteristics associated with each class, compared with classifiers trained according to the traditionally load balanced traffic. In cases that the classes are represented by a very small portion of the traffic, such as low frequent attacks, the impact is even greater. If such traffic is traditionally load balanced, there may not be enough information on each instance to properly characterize that class.

As groups of similar flows are naturally uneven in size, the first and the second goals will nearly always be contradictory. Thus, it is required for the CLB to use a clustering algorithm capable of forming balanced cluster sizes. This problem is known as *balance-driven clustering* or *balance-constrained clustering* [MF14]. The former allows some level of imbalance while the latter requires a perfect balance.

Chapter 2

Previous Work

Previous works present clustering and flow correlation methods for NIDS. However, none of which were used for scale-out by LB as presented in this paper. They either propose load balancing approaches for non-learning NIDS or use clustering to improve ML classification performance but not for load balancing (e.g., [LKT15]).

To the best of authors' knowledge, this is the first work that addresses scaling out of ML-based NIDS by LB. The following subsections describe approaches for balance driven clustering from different fields of computer science and their applicability to our problem.

2.1 Graph Partitioning

In graph theory the problem is known as *uniform/balanced graph partitioning*. It deals with partitioning a graph to k equal-size blocks of nodes, such that the weights on edges across blocks are minimized or maximized. This problem is known to be NP-complete [BMS$^+$16]. In this domain, our problem can be formulated as a graph with network flows represented by vertexes, distances between flows (similarity metric) represented by weights on edges and k equals number of NIDS instances. Given a solution, every graph partition forms a cluster of similar network flows. The k partitions (clusters) are promised to have near equal sizes. Figure 2.1 demonstrates such partitioning.

Since the problem is NP-complete, practical solutions use heuristics. These can be categorized into several broad approaches. Local approaches such as Kernighan–Lin [KL70]

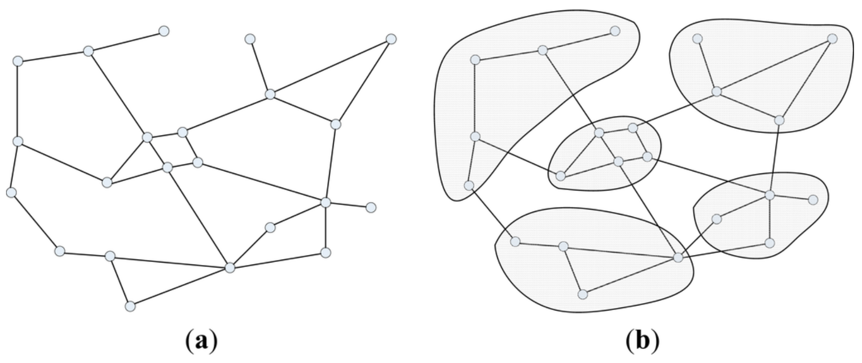


Figure 2.1: (a) network traffic modeled as a graph, vertexes represent flows, edges represent flows similarity. (b) balanced partitioning example.

and Fiduccia-Mattheyses [FM88] iteratively improve the starting solution based on local information. Their main drawback is low quality partitioning resulting from an arbitrary initial step. Global approaches such as spectral partitioning [HL95] find solutions based on the properties of the entire graph. However, they are often used for small graphs as their time complexity incur scalability issues [BMS$^+$16]. The most successful approaches for large graphs are based on multilevel graph partitioning. These approaches reduce the size of the graph, partition the reduced graph, map it back to original graph and refine it. One famous example is the algorithm proposed by Karypis and Kumar, *multilevel k-way partitioning*, implemented in METIS [KK98]. This algorithm is evaluated as a CLB.

2.2 K-means Variations

K-means is a well known clustering algorithm with no balancing constraint. In the following, we review K-means variations that impose a balancing constraint either on every algorithm iteration or as a post-processing step.

[BBD00] propose K-means variation that promises minimal cluster size by formalizing a linear algebra optimization problem with constraints on minimal cluster size. On every iteration, the cluster assignment step is solved as a minimum cost flow problem using linear programming or network simplex. However, the time complexity of $O(n^3)$ makes this algorithm impractical for a network load balancer.

Similarly, [CNMY14] formalize balanced K-means as linear algebra optimization problem. The balance is achieved by minimizing both the sum of squared distances between data points and centroids (objective of K-means) and the squared sum of number of data points in each cluster (exclusive lasso regularizer). The relative weight of the competing variables in the objective function is controlled via parameter. This algorithm is evaluated as a CLB.

[LHNL17] propose similar method using least square linear regression and augmented lagrange multipliers. The linear regression estimates the hyperplanes that separate one class from another. Augmented lagrange multipliers method is used to estimate a solution to the optimization problem. The algorithm's time complexity is $O(n^2c + d^2c)$ and the space complexity is $O(n^2 + d^2)$ (n - number of data points, d - dimensions, c - number of centroids); hence, the evaluation of large datasets is not feasible.

[MF14] propose balance-constrained K-means. The algorithm is essentially K-means with a different assignment step. Instead of assigning data points to their closest centroids, they are assigned to one of n slots, where every n/k slots are preallocated to a cluster. This assignment problem formalization is solved using the Hungarian algorithm. Its time complexity of $O(n^3)$ makes it impractical for a network load balancer as well.

[GCC14] propose simple K-means variation where data points are assigned to closest centroid only if its cluster size is less than n/k. This means that clustering quality highly depends on centroids initialization. Due to this, the authors assume prior knowledge of at least few data points per each cluster is available, which is not the case in our problem.

[YMC11] proposes *MCBC*, a K-means variation that assigns each data point to either nearest cluster to decrease objective function or to nearest constraint-unsatisfactory cluster to improve balancing. Its time complexity is not analyzed, though it is clear that in the worst case it is $O(n^2)$ which doesn't scale well. Empirical evaluation shows that it is indeed the case. Moreover, clusters that are bigger than n/k have tendency to get wrapped by other clusters as their outermost data points get reassigned (figure 2.2). This behavior is undesired for our problem.

[HXSZ09] proposes to run K-means as is and then iteratively adjust the borders between the resulting clusters. The algorithm describes how to find bordering data points. However, it doesn't explain how to adjust the borders nor when the process converges. Therefore it

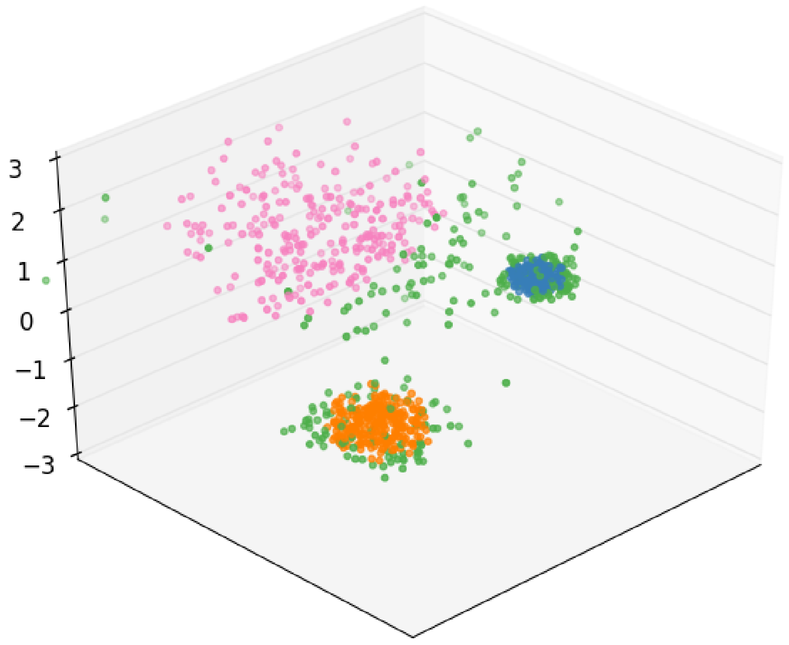


Figure 2.2: Clustering result of *MCBC* algorithm. The green cluster wraps the other ones.

is not evaluated.

2.3 Other Algorithms

Agglomerative clustering methods can be adapted to achieve balanced clusters: once a cluster reaches a certain size in the bottom-up agglomeration process, it can be removed from further consideration. However, this may significantly impact cluster quality [BG06] and since its complexity of $O(n^2)$ does not scale well, it is inapplicable for our problem.

[ZWL10] proposes generic approach that gets as input any clustering result (assignment of every data point to a cluster) and a list of constraints such as data points that must/must not be on the same cluster as well as clusters' sizes. The algorithm returns new clusters assignment that satisfy the constraints while having minimal deviation from input clusters assignment. It is done by formulating an integer linear programming optimization problem that maximizes agreement between partitions. It may be used as a CLB if size constraints of n/k are provided. However, since it doesn't take into account distances between data points, we expect that the clustering quality will be lesser of the aforementioned methods and therefore it is not evaluated.

Chapter 3

Clustering Load Balancer

In the below, we propose different methods for clustering based load balancing. Detailed evaluation of their load balancing performance and how they affect the ML-based NIDS is presented in chapter 4.

The following methods are based on existing off-the-shelf balance-driven clustering algorithms.

1. **Multilevel K-way graph partitioning (Multipart).** The graph we build consists of n vertexes, each represents a network flow. The distance metric we use is Euclidean distance. In order to reduce run time and memory footprint, we don't use a complete graph, which have $\binom{n}{2}$ edges. Instead, we only use distances between data points and their nearest 100 neighbors, which have upper bound of $100n$ edges. We evaluate the algorithm implemented in `METIS`, using the `metis-python` wrapper [Wat17]. This implementation can only minimize the objective function; i.e., weights across partitions. However, our objective is that close data points in the Euclidean space will end up on the same partition whereas distant data points will end up on different partitions. To achieve that we scale all distances to a fixed range $[0, 1000]$ and use $1000 - \textit{distance}$ as weights.

2. **Balanced K-means using exclusive lasso regularizer (XLK-means).** We implement the algorithm from [CNMY14]. It requires that on every iteration, for each data point and centroid, the following expression will be calculated:

$||X - HF^T||_F^2 + \gamma Tr(F^T 11^T F)$ where $X \in \mathbb{R}^{D \times N}$ is the data points (network flows) and $F \in \{0,1\}^{N \times C}$ is clusters assignment matrix (N data points, D dimensions, C clusters). Naïve implementation that recalculates the full expression would have been slow. Our implementation calculates for each data point only the expression parts that can actually change, while keeping intermediate results throughout the iteration. In addition, we avoid the inner loop (going over each centroid) using matrix multiplications and reduce convergence time by adopting inertia based convergence criteria. All these improvements result in significant speed up. For evaluation we use $\gamma = 0.1$ which provides satisfactory empirically results.

In the below we propose new methods for clustering based load balancing. We use the K-means clustering algorithm in different ways to achieve balanced clustering.

3. **K-means on subset of features (K-means)**. This is the main concept we present in our paper [FBIM17]. We search for subsets of features with specific preprocessing options (encoding and scaling) such that when clustered by K-means, the result is relatively balanced. Table 3.1 presents the predefined features subsets that we search through. They were chosen manually. The evaluation framework presented on section 4.1.3 filters out imbalanced clustering results; thus, leaving in only features subsets that naturally result in balanced clusters.

Clustering Feature Sets	#	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	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 3.1: Features options for traffic clustering. Each row describes a predefined subset of features. The column with hash sign denotes number of features.

4. **K-means with greedy clusters merge (KC-means)**. We run K-means with $K = kc$, where k equals to the requested number of clusters and c is a constant.

The resulting kc clusters need to be merged into k output clusters. If only cluster sizes are considered (without inter-cluster distances), this problem is equivalent to the NP-hard problem of *balanced multi-way number partitioning*. We solve it by sorting the kc clusters according to size in descending order and assigning them one by one to the smallest output cluster. This greedy heuristic may not find an optimal solution but its run time, $O(kc \log(kc))$ is acceptable. Other heuristics are discussed in [ZMP11]. This approach can be generalized to any clustering algorithm that gets k desired number of clusters as an input. However, the level of imbalance is uncontrolled and if one of the kc clusters is significantly larger than n/k, the output k clusters will be highly imbalanced, regardless how they are merged.

For evaluation we use $c = 5$, an arbitrary choice. Our experiments show that this constant has no significant impact on the results as long as it is big enough.

5. **Weighted K-means (WK-means)**. We attempt to achieve balanced K-means result by weighting every feature in proportion to its level of balance; i.e., how evenly its values are spread across their range. Features that are more balanced will get higher weights; thus, the probability of having balanced clustering result will be higher. Unfortunately, K-means doesn't support features weighting. However, the desired effect can be achieved by preprocessing every feature's variance (Appendix A). We standardize every feature to a unit variance. Then, we assign it a weight of $(\sigma(D)/\mu(D))^{-4}$ where D is the differences between consecutive values. We use the well known relative standard deviation σ/μ which is a standardized measure of dispersion to quantify the level of imbalance of every feature. The exponent in the formula amplifies weight's magnitude.

Chapter 4

Evaluation

In this chapter, we evaluate our approach; i.e., the improvement gained by using clustering load balancer for misuse ML-based NIDS. In section 4.1 we describe our evaluation setup (i.e., the traces, models, tools and method). Then, in section 4.2 we present the results as follows. Section 4.2.1 compares the CLB approaches in terms of clustering quality and run times. Section 4.2.2 demonstrates flows distribution of traditional LB models and CLB approaches. Section 4.2.3 presents ML performance comparison of centralized model, traditional LB models and CLB approaches. Lastly, in section 4.2.4 we discuss the scalability of the CLB approaches.

4.1 Evaluation Setup

4.1.1 Traces

For evaluation, the well known NSL-KDD [TBLG09] dataset is used. It is reduced version of KDD CUP '99 [SFL$^+$00], more adequate for machine learning benchmarks. This dataset has 148,517 records based on traffic captured during nine weeks simulation of military network under attack. The data from the first seven weeks of the simulation is used for the training of the models and the data from last two weeks which includes new types of attacks is used for the testing.

Each record is made of 41 features derived from a sequence of packets in a TCP session.

The features can be divided into few categories: basic information of the individual TCP connection (e.g., duration, source bytes), expert knowledge, statistics over all connections to the same destination host/service in the last two seconds and statistics over last 100 connections to the same host/service.

Records are labeled as either *Normal* or one of four classes of attack: *DOS* (Denial of Service), *PROBE* (scanning), *R2L* (Remote-to-Local, unauthorized remote access), or *U2R* (User-to-Root, unauthorized local access).

4.1.2 Load Balancer Models

Three types of models are evaluated as load balancers: baseline (centralized), traditional LB and clustering LB. Models get traffic and k number of instances as input and return data slices as output. Every model consists of a load balancing algorithm as detailed in table 4.1 and data preprocessing options as detailed in table 4.2. The clustering LB model that evaluates vanilla K-means has in addition a subset of features. Models' quality is measured in the following aspects:

Misuse Detection Performance. That is the performance of the ML classifiers trained and tested on NIDS' instances. We evaluate classification algorithms that are commonly used for misuse detection, namely, 1-Nearest-Neighbor (1NN), Decision Tree (DT), Random Forest (RF), Multilayer Perceptron (MLP) and support vector machines (SVM). The performance metrics we use are precision, recall, F-score and area under curve (AUC).

Model Type	Algorithms	Short Name
Baseline (centralized)	All traffic is processed by a single NIDS instance	Baseline
Traditional LB	Round Robin flows distribution	Round Robin
	Uniform random flows distribution	Uniform Random
Clustering LB (detailed in chapter 3)	Multilevel k-way graph partitioning	Multipart
	Balanced K-means using exclusive lasso regularizer	XLK-means
	K-means on subset of features	K-means
	K-means with k=rc and greedy clusters merge	KC-means
	Weighted K-means	WK-means

Table 4.1: Load balancing model types and evaluated algorithms

Preprocessing aspect	Evaluated options
Data scaling	No scaling Range scaling according to min/max values Standard scaling - data centered around mean with unit variance
Encoding of categorical features	No categorical features, only numeric One-hot encoding

Table 4.2: Data preprocessing options used for LB models evaluation

Load Balancing. Two metrics determine the level of balance: σ_k - standard deviation of slice sizes and MMR_k - ratio between maximal and minimal slice sizes. Model's level of balance is defined as $\sigma = \max_k(\sigma_k)$, $MMR = \max_k(MMR_k)$.

Run Time. We measure the absolute time it takes the load balancing algorithms to process all input flows, in seconds. As the evaluated algorithms are not optimized for production workloads, nor for the hardware we used, the measured run times are not comparable with a real life system. Instead, they give a sense of relativity between the clustering algorithms.

Scalability. In today's dynamic nature of traffic, it is important that the clustering load balancer would work for any k. To this end, we analyze the empirical correlation between k and misuse detection performance, load balancing quality and run time.

4.1.3 Tools and Method

The evaluation framework is based on Python 2.7 and the scikit package[1]. It ran on Amazon Linux 64-bit, Intel Xeon E5-2666 v3 @ 2.9GHz, 7.5GB RAM. The source code is publicly available at `https://github.com/frishrash/thesis_code`.

The evaluation process is depicted in figure 4.1. First, the dataset is split to train and test sets. We start with training data. Every LB model (which consists of LB algorithm, preprocessing option and optionally features subset) is evaluated on varying number of instances $k = \{3..9\}$. Load balancing time is recorded. We define a load balance feasibility criteria: $\sigma \leq 6500, MMR \leq 20$. Only models that meet this criteria continue to next

[1]a free software machine learning library for the Python programming language [PVG+11] (`http://scikit-learn.org`).

evaluation step. Infeasible models are discarded. The thresholds were chosen such that most of models were deemed infeasible, yet, there was enough variety among the feasible ones.

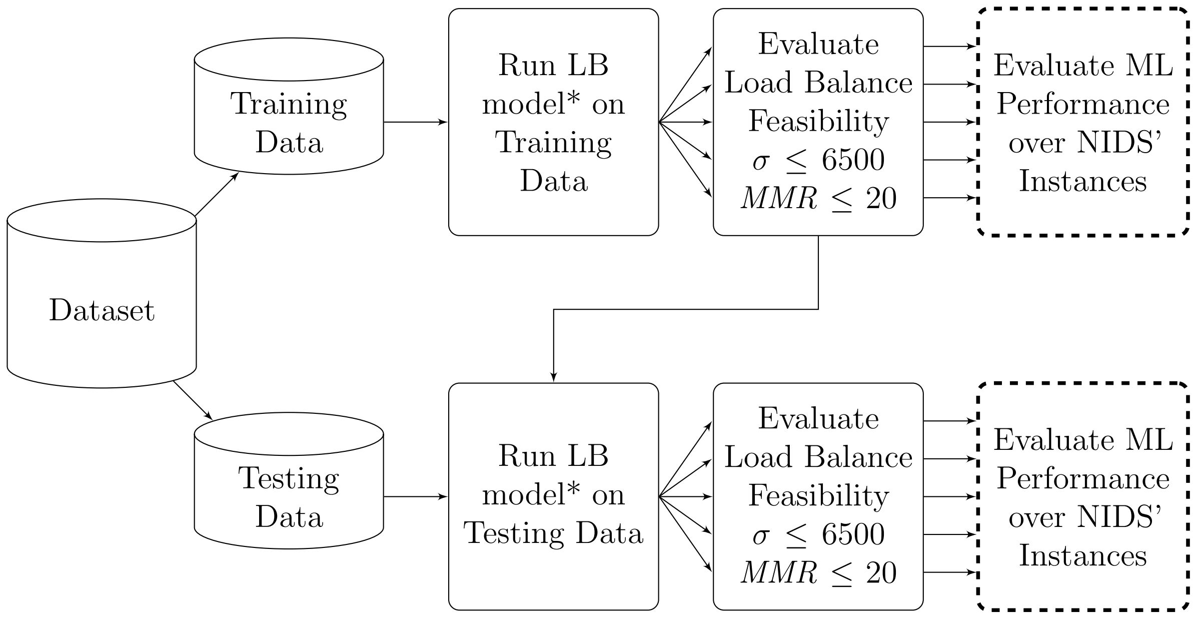

Dataset
Training Data
Testing Data
Run LB model* on Training Data
Run LB model* on Testing Data
Evaluate Load Balance Feasibility $\sigma \leq 6500$ $MMR \leq 20$
Evaluate ML Performance over NIDS' Instances

Figure 4.1: Workflow of load balancing model evaluation (* - using the *same* model)

The next step evaluates misuse detection performance of the LB model for every classification algorithm (1NN, DT, RF, MLP, SVM[2]). Figure 4.2 depicts the evaluation process for a single algorithm. Each data slice is assigned to NIDS instance. Classification algorithm's performance is evaluated using 5-fold cross validation on all instances in parallel. The confusion matrices and prediction probabilities from all instances are summarized into a single matrix from which performance measurements are calculated.

Models that complete evaluation on the training data repeat the same process on the testing data. By the end of evaluation process, only valid models are left. For each, the framework records {LB model, preprocessing options, features subset, dataset, number of instances (k), load balancing time, level of balance (σ, MMR), classification algorithm, misuse detection performance}.

[2]SVM is evaluated only on min-max scaled data due to excessive run times.

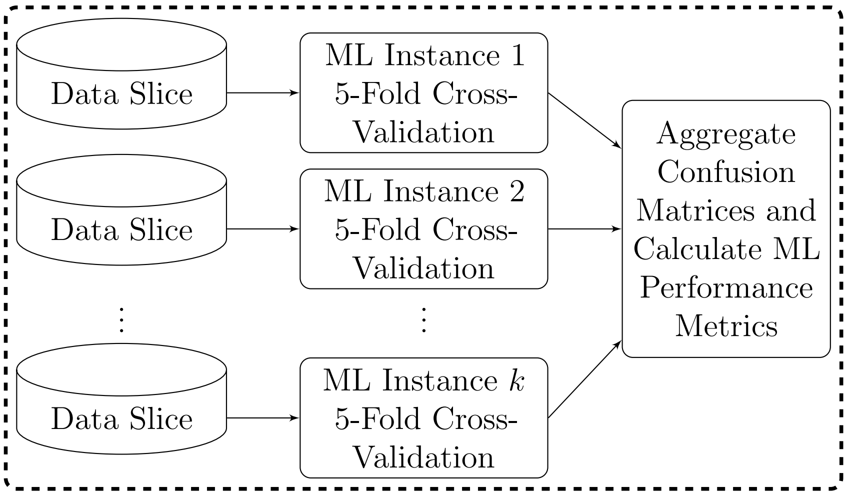

Data Slice
ML Instance 1 5-Fold Cross-Validation
Data Slice
ML Instance 2 5-Fold Cross-Validation
⋮
⋮
Data Slice
ML Instance k 5-Fold Cross-Validation
Aggregate Confusion Matrices and Calculate ML Performance Metrics

Figure 4.2: Workflow of ML performance evaluation over NIDS' instances

4.2 Results

4.2.1 Load balancing performance of the CLB models

In this subsection, the quality of all feasible CLB models is discussed in terms of run times, σ and *MMR*. Table 4.3 presents all feasible CLB models and compares the best performing model of every clustering algorithm. Feasible models are marked with ✓. Best performing model is highlighted in red ✓ and its performance from the testing data is presented.

The scaling and encoding refer to data preprocessing options as presented in table 4.2. NU stands for numeric only encoding and OH stands for one-hot encoding. Evaluated clustering algorithms are the ones presented in table 4.1 using short names. Rectangular

Clustering	Min-max		None		Standard		Best Model Performance		
Algorithm	OH	NU	OH	NU	OH	NU	Avg. Run-time	σ	*MMR*
Multipart		✓	✓	✓	✓	✓	9.17s	174.3	1.06
XLK-means	✓	✓			✓	✓	6.7s	4.92	1.01
KC-means	✓	✓			✓	✓	5.58s	512.09	1.62
K-means	✓	✓					3.25s	5358.91	8.11
K-means [3]	✓	✓	✓	✓			1.77s	2643.96	14.9
K-means [5]	✓						3.53s	3452.61	7.55
K-means [7]	✓	✓	✓	✓			2.39s	5731.14	11.52
WK-means	✓	✓	✓	✓	✓	✓	3.08s	6472.2	19.81

Table 4.3: Feasible CLB models and performance of best model per clustering algorithm. OH stands for one-hot encoding, NU for numeric only. σ and *MMR* determine standard deviation and min-max ratio of cluster sizes as defined earlier.

brackets refer to a features subset as presented in table 3.1.

For every clustering algorithm, the best performing CLB model is compared with the others in terms of run times, σ and *MMR*.

To summarize, the fastest clustering algorithm is K-means. However, sizes of resulting clusters are far from being perfectly balanced. WK-means run time is similar but its balancing performance is worse. Next, KC-means is a bit slower but shows a great increase in balancing performance. XLK-means is somewhat slower than KC-means but its balancing performance is near perfect. Lastly, Multipart is the slowest but with balancing performance that is only second to XLK-means. It is important to stress that although all models were evaluated on the same framework, run times are also depend on the concrete implementations and may be significantly improved. For example, scikit's implementation of K-means algorithm uses compiled native code, while XLK-means is implemented in pure python.

Multipart. For all feasible models, run time behaves similarly. That is, for $k = \{4..9\}$ run time on training data is between approx. 8.5s and 10s and on testing data between approx. 7s and 8.7s. However, for $k = 3$ run times are extremely higher, with the model of *None-NU* being the fastest (21.7s on training data, 18.3s on testing data) and the model of *Standard-OH* being the slowest (106.3s on training data, 86.7s on testing data). We couldn't find explanation for this phenomenon, we assume it is a property of the algorithm or its implementation in `METIS`. In terms of balancing, all models have the same *MMR* value of 1.06. σ varies with no apparent correlation to k or to the model (same model behaves differently on training and testing data). Since absolute σ values are relatively low (73-225 on training, 83-174 on testing), best performing model is chosen mainly according to run time, which is *None-NU*.

XLK-means. Results show strong correlation between run times and level of balance; i.e., fastest model is also the most balanced and vice versa. This surprising behavior may be explained by the fact that run time is mostly dominated by number of iterations until convergence. We assume that some models, without the regularizer are naturally closer to a balanced form and therefore converge faster and yield a more balanced clusters. There is no correlation between run time/balance and k. The models ordered from best to worst are: *Min-max-NU*, *Min-max-OH*, *Standard-NU*, *Standard-OH*. Run times vary from 7.29s to 47.38s on training data and from 6.7s to 39.98s on testing data. *MMR* varies between 1.01 and 1.09 and σ varies between 7.45 and 75.43 on training data and 4.92 to 62.66 on testing data.

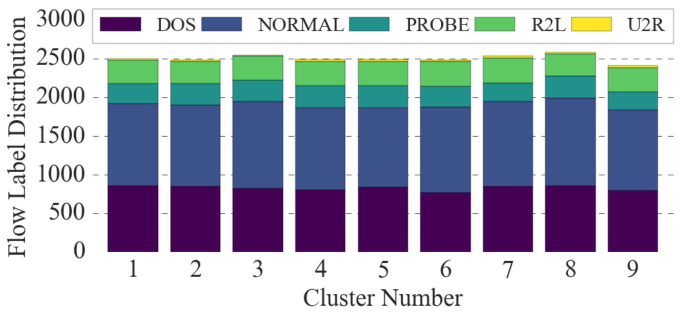
KC-means. For this algorithm, run times increase almost monotonically with k since merging time is dominated by kc. At large, models with numeric encoding run faster than models with one-hot encoding at the expense of yielding less balanced clusters. Run times vary between 5.66s and 8.94s on training data and between 5.48s and 8.32s on testing data. *MMR* values are best for models with min-max scaling, ranging from 1.34 to 1.76 and worst for models with standard scaling, ranging from 2.61 to 4.91. Similarly, σ values range from 323 to 855 on models with min-max scaling and from 2145 to 2865 on models with standard scaling.

K-means. Run times are mostly dominated by number of features and in general range between 1.77s and 4.64s across different features subsets and training/testing datasets. *MMR* values range between 7.55 and 17.75. σ ranges between 2643.96 and 5731.14.

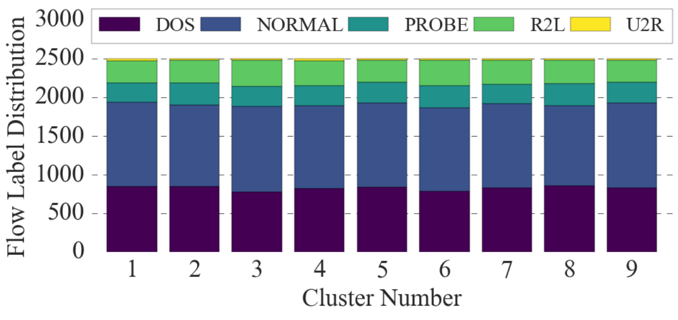
WK-means. Since this algorithm has its own features normalization method, run times are not affected by model scaling. However, they are affected by model encoding since models with numeric encoding disregard the categorical features, which means less features to work on. On training data, run times are around 3.8s for models with numeric encoding and around 5.3s for models with one-hot encoding. On testing data, run times are 3.1s and 4.8s accordingly. *MMR* values are nearly the same for all models. On training it is approx. 19.4, on testing 19.8. Similarly, σ values are nearly the same for all models, around 6080 on training and 6475 on testing.

4.2.2 Flows Distribution

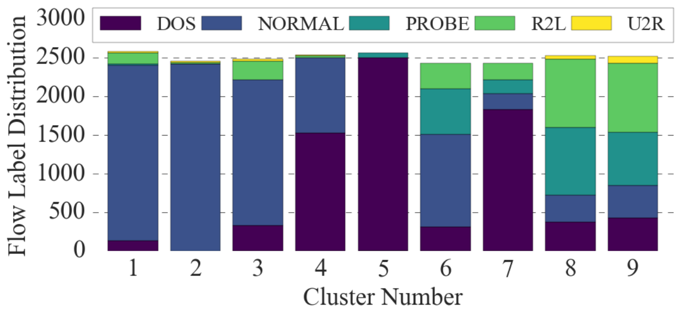
CLB models are expected to group similar network flows (assumed to be of the same class) to the same cluster. An ideal clustering result would maximize clusters' homogeneity; i.e., classes that have less than n/k data points should be on a single cluster and classes that have more than n/k data points should be spread over minimal number of clusters, yielding mostly homogeneous clusters. In such ideal case, NIDS instances that process the homogeneous clusters will achieve perfect classification as all their data points belong to a single class. Instances that process the non-homogeneous clusters may achieve better classification than if they processed random traffic since they should have more data points belonging to the same class. The misuse detection performance will be discussed in the next subsection but to demonstrate that CLB models indeed group similar network flows, figure 4.3 presents network flows' class distribution for traditional LB models, Multipart and XLK-means CLB models for $k = 9$. As can be seen, some clusters of the CLB models are close to being homogeneous.



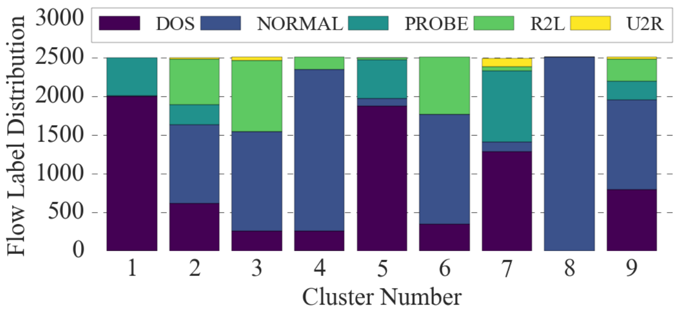
(a) Flows distribution of Uniform Random.



(b) Flows distribution of Round Robin.



(c) Flows distribution of Multipart.



(d) Flows distribution of XLK-means.

Figure 4.3: Flows label (class) distribution across clusters, for $k = 9$. On top - traditional LB models, on bottom - CLB models.

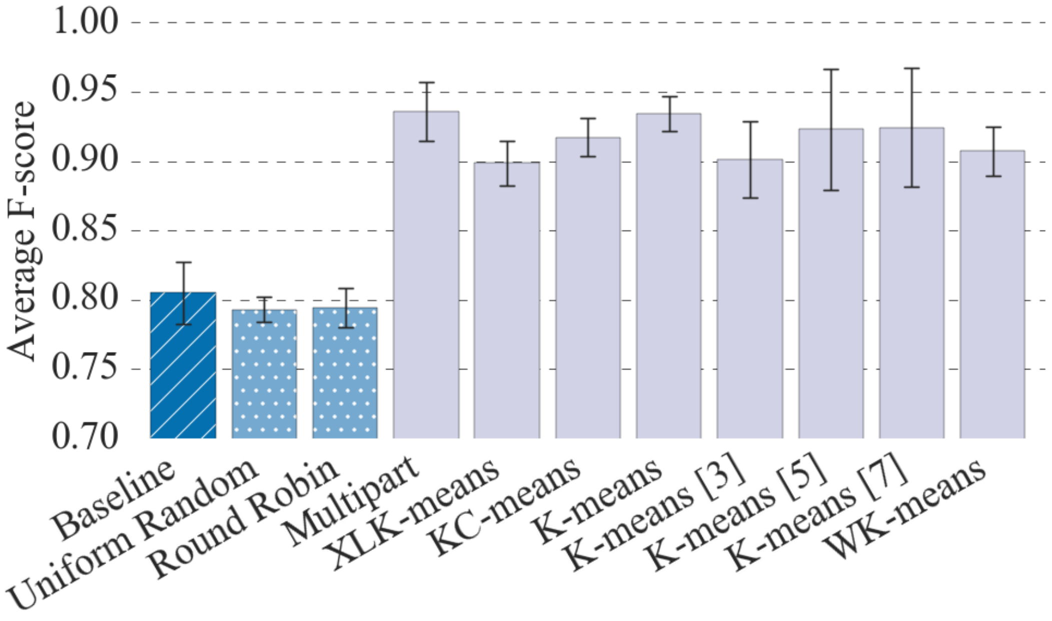
4.2.3 NIDS misuse detection performance

In this subsection, the LB models (table 4.1) are compared for their resulting misuse detection performance for every classification algorithm. There are 48 feasible LB models (30 clustering, 12 traditional, 6 baseline), each model is evaluated on 5 different classification algorithms, on 7 different k's, on training and testing data, and for each there are 7 different evaluation metrics (AUC per class, precision, recall). This yields more than 20,000 individual results. In order to present and compare the results we filter and aggregate them as follows: (1) Only results on testing data are presented. (2) As precision and recall results are very similar we only present their harmonic mean, known as F-score. (3) For every model, F-score and AUC is averaged over all K's. (4) Among the CLBs, only the best performing models, in terms of clustering quality and speed are compared. (5) When several models of the same LB approach/algorithm have similar results, evaluation metrics are averaged over these models as well.

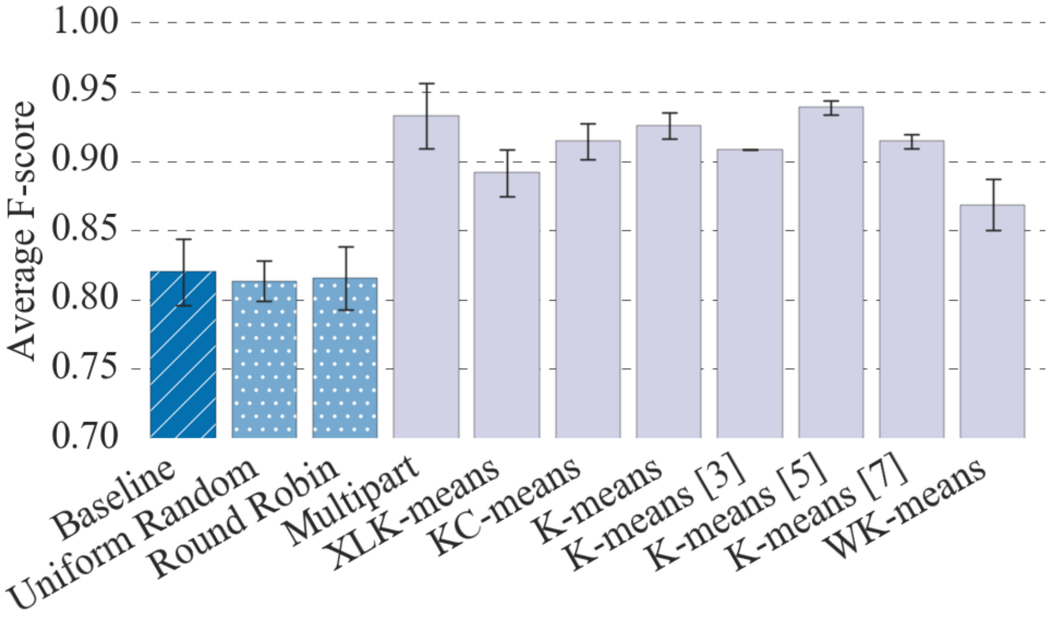
1-Nearest-Neighbor. Average F-score of baseline models is 0.97. Traditional LB models are near 0.95. CLB models vary between 0.95 and 0.97. While all CLB models perform better than the average traditional LB model, the improvement is insignificant. However, the insignificance is only due to the very high score that traditional LB models achieve using 1-nearest-neighbor algorithm on NSL-KDD dataset. Area under curve follows a similar pattern, where average AUC of every CLB model is greater than the average AUC of traditional LB models by 0.1-0.5. The average AUC of best performing CLB model is near equal baseline's.

Decision Tree and Random Forest. For these classification algorithms, the improvement of CLB models over traditional LB and baseline models is the most prominent. Figure 4.4 presents the average F-score and standard deviation for every LB model. Using decision tree classification algorithm, the average F-score of CLB models is between 5.92% and 15.36% greater than rest of models (WK-means vs. baseline and K-means [5] vs. Uniform Random accordingly). In case of random forest, the improvement is even greater. **CLB models perform between 11.62% and 18.01% better than rest of models** (XLK-means vs. baseline and Multipart vs. Uniform Random accordingly).

The average AUC using decision tree and random forest classification algorithms for all CLB models is greater than traditional LB's. Vast majority of these models also beat the baseline. As an example, figure 4.5 depicts ROC plots with AUC of classes *NORMAL*, *U2R* and *R2L*, for all models when $k = 9$.

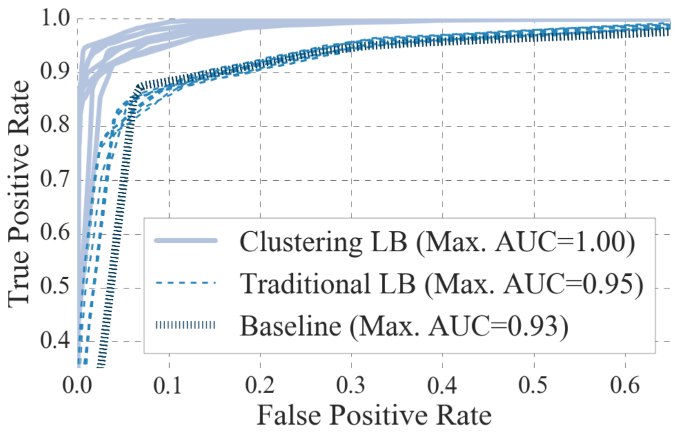


(a) Results for random forest classification algorithm.

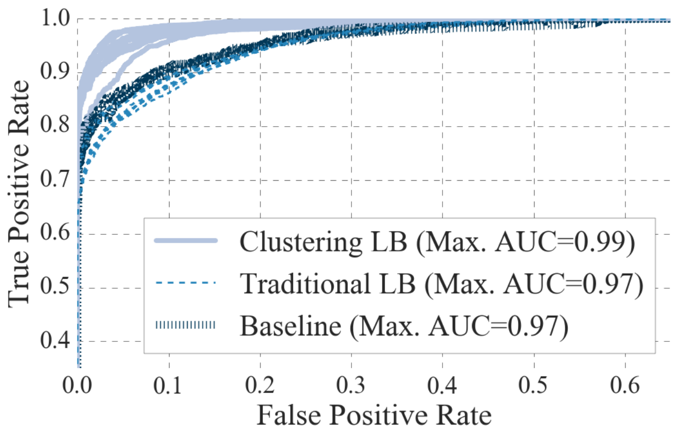


(b) Results for decision tree classification algorithm.

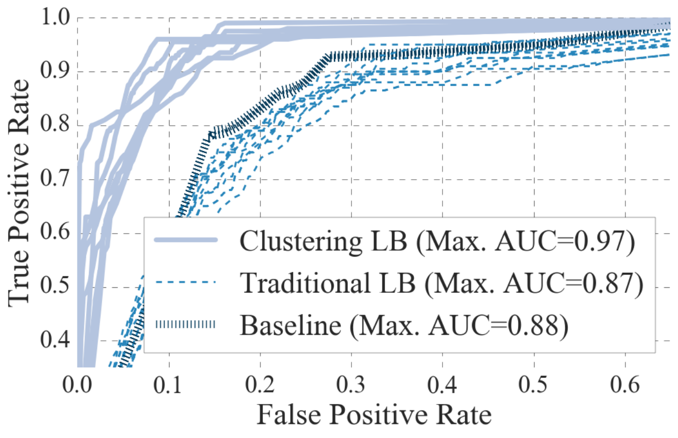
Figure 4.4: Average F-score and standard deviation of every LB model using decision tree and random forest classification algorithms.



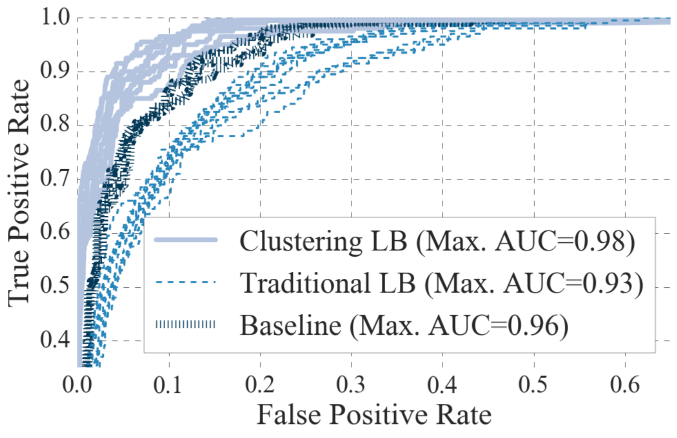
(a) Decision tree classifier, class *NORMAL*.



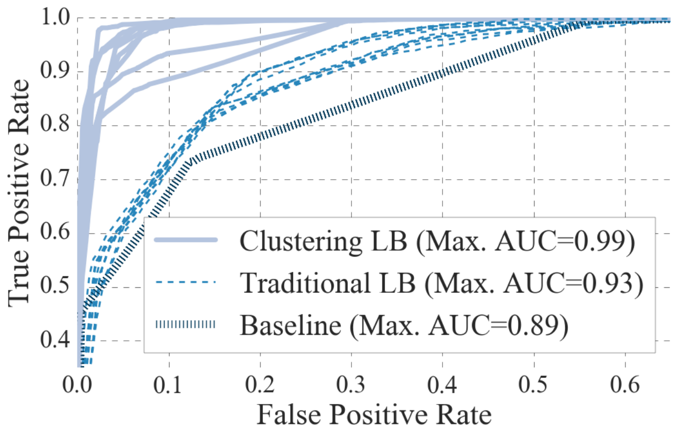
(b) Random forest classifier, class *NORMAL*.



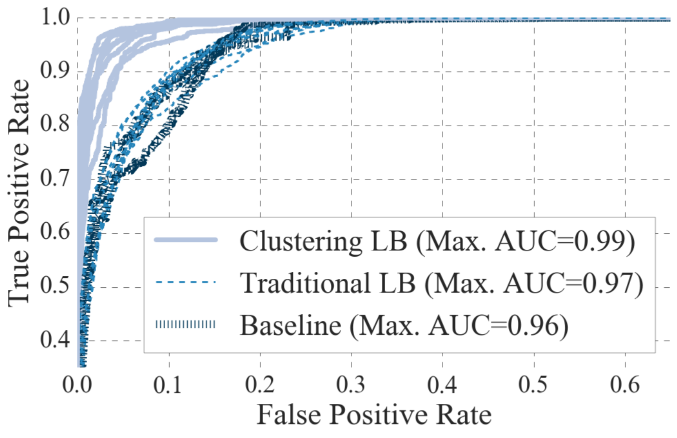
(c) Decision tree classifier, class *U2R*.



(d) Random forest classifier, class *U2R*.



(e) Decision tree classifier, class *R2L*.



(f) Random forest classifier, class *R2L*.

Figure 4.5: ROC plots for classes *NORMAL*, *U2R* and *R2L* on testing data, $k = 9$ of decision tree and random forest classification algorithms.

The performance improvement of CLB models over the baseline is quite surprising as one might assume that since all traffic is processed on a single NIDS instance, it should outperform any LB model using the same classification algorithm. However, this assumption is flawed as the same complexity constraints are used both for the baseline and for each and every one of the instances. More concretely, let d denote the maximal depth constraint we impose on our decision trees/random forests and n number of network flows we use for training. For the baseline model we have a single decision tree of depth d built during training over n flows. For every LB model we have k decision trees (one per instance), each of depth d built during training over n/k flows with the LB effectively employing additional tree level. Such comparison is valid from a practical point of view, e.g., in cases centralized NIDS appliance is replaced by a CLB. However, the underlying classifiers are not directly comparable and should not be expected to have the same misuse detection performance.

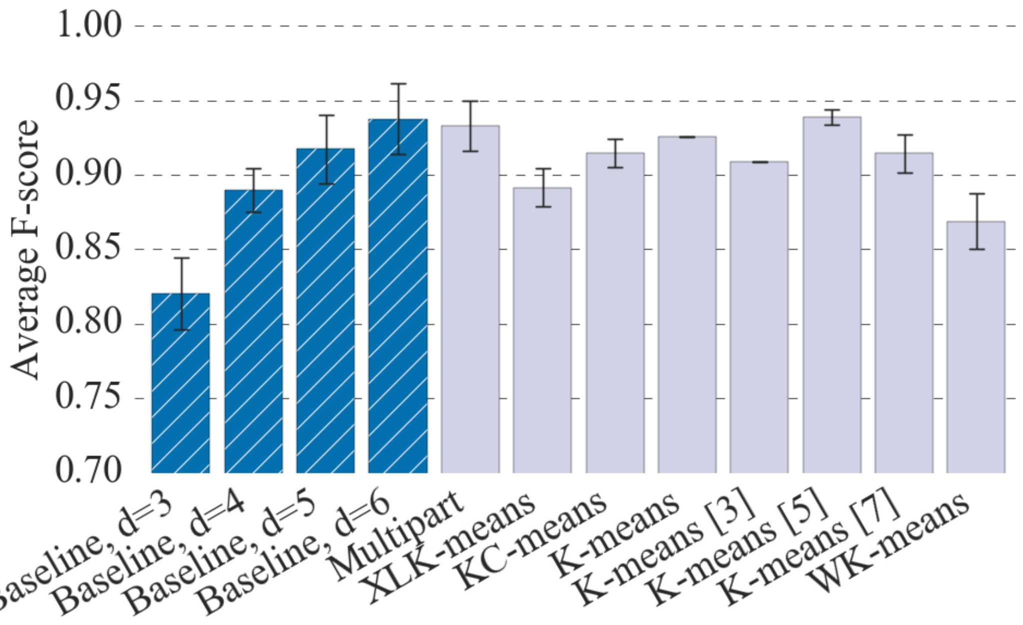
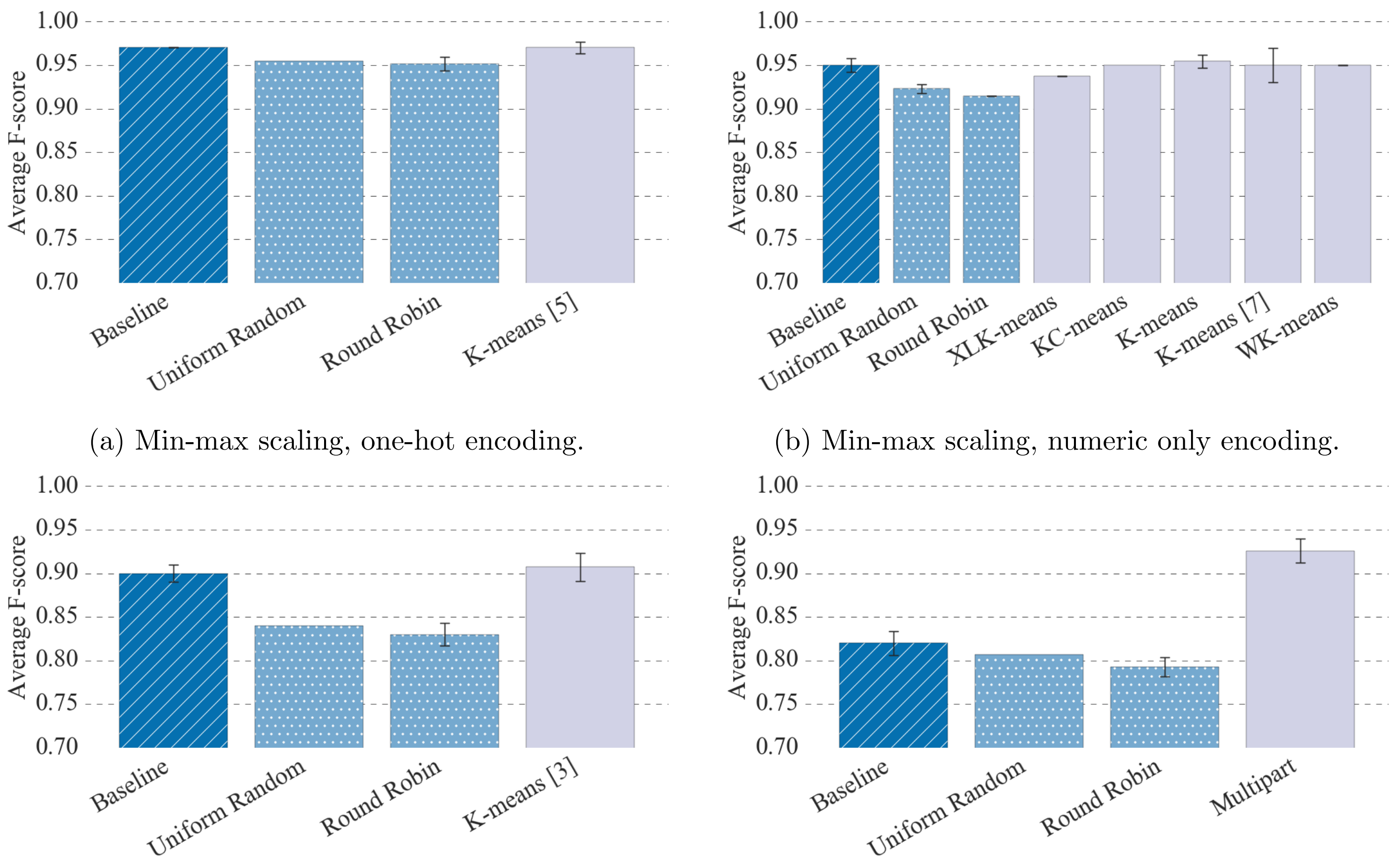


Figure 4.6: Average F-score of baseline model with varying decision tree depths compared with CLB models with decision tree of depth 3.

We can however quantify the decision tree depth for which baseline's performance would be comparable with the CLBs. To this end we evaluate the baseline model with decision trees of varying depths. Figure 4.6 compares the average F-score of baseline models with $d = 3..6$ and the CLB models (using decision trees of depth 3). It can be seen that Multipart and K-means [5] are equivalent to a baseline of 6 levels, KC-means and rest of K-means models are equivalent more or less to a baseline with 5 levels, XLK-means to a baseline of 4 levels and WK-means somewhere between 3 and 4 levels.



(a) Min-max scaling, one-hot encoding.

(b) Min-max scaling, numeric only encoding.

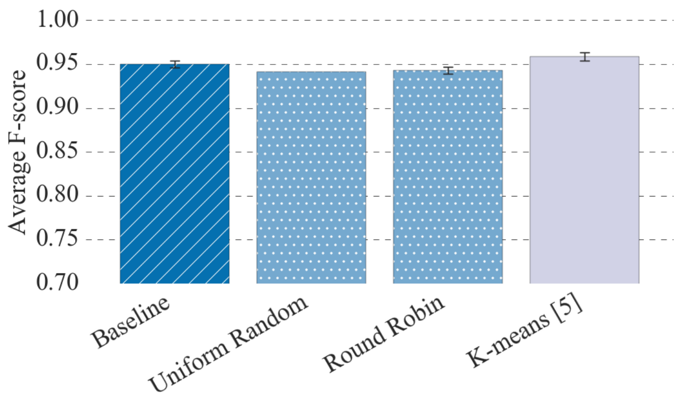
(c) No scaling, one-hot encoding.

(d) No scaling, numeric only encoding.

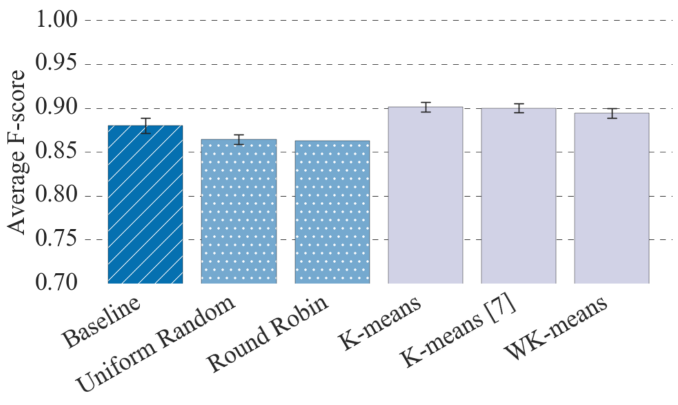
Figure 4.7: Average F-score of LB models using MLP classification algorithm per encoding and scaling.

Multilayer Perceptron. The results of this classification algorithm greatly depend on models' encoding and scaling options. Since F-score variation of baseline and traditional LB models is high, we compare the LB models per encoding and scaling as presented in figure 4.7. Standard scaling is not presented since none of the CLB models use this scaling option. As can be seen, in all cases the average F-score of the CLB models is better than the average F-score of traditional LB models. For models scaled according to minimal and maximal values this improvement is insignificant as the scores of all LB models are high. For models without scaling the improvement is more significant and especially in the case of numeric only encoding where average F-score of Multipart is 0.93, baseline's is 0.82 and traditional LB's is 0.8. AUC follows very similar behavior for most classes and CLB models. For all, the average AUC of the CLB models is either the same or better than the rest. The most significant improvement achieved in the case of models without scaling and numeric only encoding, for the *U2R* class, with average AUC of the CLB model (Multipart) 0.88 where baseline's is 0.83 and traditional LB's around 0.75.

SVM. The evaluation of SVM classifier has several limitations. First, we evaluate only models scaled according to minimal and maximal values due to excessive run times of SVM on other scaling options. Second, the results are sensitive to encoding. The end result is that only 4 CLB models complete evaluation using SVM. The average F-score of all LB models per encoding is presented in figure 4.8. Average AUC of all CLB models is nearly the same as baseline's and traditional LB's for all models and classes.



(a) Min-max scaling, one-hot encoding.



(b) Min-max scaling, numeric only encoding.

Figure 4.8: Average F-score of LB models using SVM classification algorithm per encoding.

4.2.4 Scalability

To evaluate CLB models' scalability we look at run time and F-score as function of k.

Run time. The run times of all K-means based CLB models (KC-means, K-means, WK-means) are linear in number of data points, number of dimensions and k. The theoretical run time of Multipart is not clear. Empirical evaluation shows it is independent of k expect the discussed case of $k = 3$ in which run times are significantly high. For $k = \{4..9\}$ run times are fixed up to a 1.7 seconds difference. Run time scalability of XLK-means is arguable. In our implementation, the theoretic upper bound on run time is same as K-means'. However, in practice, number of iterations until convergence dominates the run time. Empirical evaluation shows that run times increase relatively linearly as function of k for models with Min-max scaling, whereas there is no apparent correlation between run time and k for models with Standard scaling. Figure 4.9 shows the run times of best performing CLB models as function of k on test dataset.

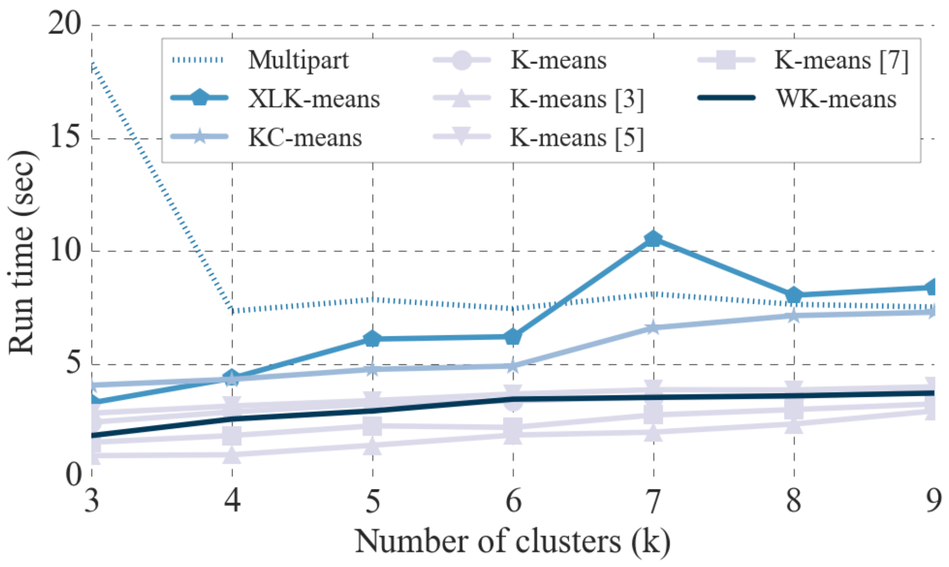


Figure 4.9: Run times of best performing CLB models on test dataset.

F-score. The F-score of misuse detection algorithms Decision Tree, Random Forest and SVM monotonically increase with k for vast majority of CLB models. For Multilayer Perceptron and 1-Nearest-Neighbors, the F-score is more or less fixed across different k's, for all CLB models. Figure 4.10 presents for each of the best performing CLB models, the average F-score across all misuse detection algorithms as function of k on test dataset.

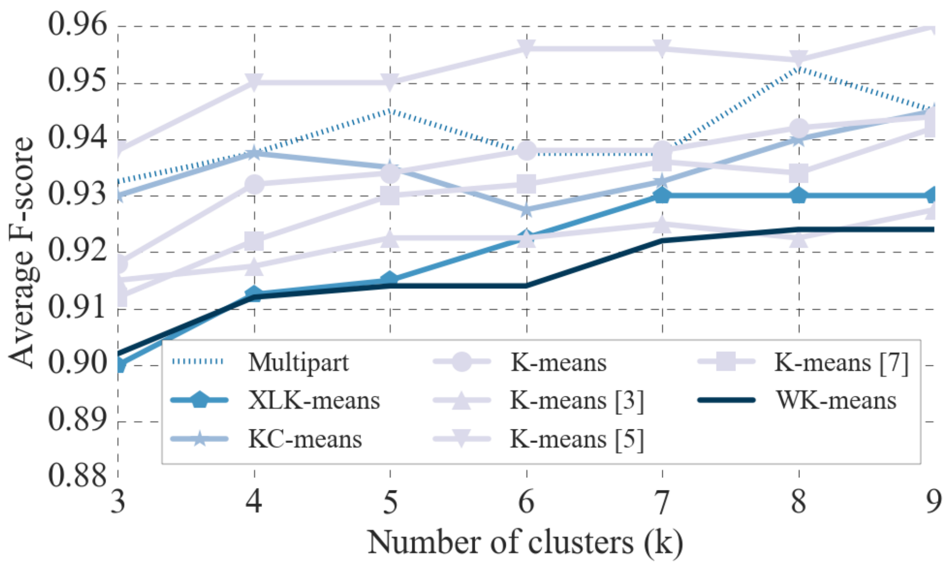


Figure 4.10: Average F-score of best performing CLB models across all misuse detection algorithms, on test dataset.

Chapter 5

Summary and Future Work

5.1 Summary

We live in an era of ever-growing network capacity demands. Data centers and enterprises deploy network capacity on demand. As the traffic bandwidth increases, the network functions, which process it, should be scaled accordingly. In particular, network security functions, for instance, ML-based NIDS. The NFV paradigm, which allows running virtual network functions on commodity hardware has enabled network operators with easy provisioning and scaling processes.

Since vertical scaling (i.e., scale-up) is limited to the capacity of a single server, a more practical approach is horizontal scaling (i.e., scale-out). That is, spawning multiple instances of the same appliance and distribute the traffic between them by using a load balancer.

However, traditional load balancers are not optimized to maximize the learning performance of ML-based NIDS. To this end, we define a new type of load balancer: the *clustering load balancer*. Its goal is to maximize the learning (misuse detection) performance while keeping a balanced load across NIDS' instances. It works by clustering the ingress traffic into groups of similar flows, while keeping groups' sizes somewhat equal. The groups are then processed independently on different NIDS instances.

We survey existing balance-driven clustering algorithms from different domains and determine their applicability for balancing traffic. Two algorithms were found applicable: Multilevel K-way graph partitioning (Multipart) and balanced K-means using exclusive lasso regularizer (XLK-means). The rest were found inapplicable due to run time complexity.

In addition, we propose three new methods based on existing algorithms. The applicable and the proposed algorithms are evaluated as clustering load balancers. We compare their run times, level of load balance and scalability. We also present and discuss the machine learning performance gain resulting from clustering load balancers compared versus traditional load balancers, and versus a centralized ML-based NIDS.

Our evaluation results show that run times and level of balance vary between the different clustering algorithms and depend on data preprocessing. K-means on subset of features is the fastest but yields moderate level of load balance; whereas XLK-means and KC-means have moderate run times but yield good levels of load balance. All clustering load balancers produce data slices that are more homogeneous as compared with traditional load balancers. Hence, using CLB outperforms the traditional load balancers in terms of machine learning performance, for all evaluated classification algorithms. The most significant performance gain is achieved by decision tree and random forest classification algorithms.

In order to allow further research and reproducing of our results, we publish our evaluation framework as open source, including the implementation of the proposed methods and the referenced algorithms.

5.2 Future Work

In this work, we present the concept of clustering load balancer for scaling of misuse detection ML-based NIDS. This work can be further extended in several directions:

1. **Scope expansion.** In this work, we focus on misuse detection, i.e., learning how to identify known attack patterns. However, there are ML-based NIDS appliances that attempt to detect anomalous traffic with respect to learned baseline (i.e., anomaly

detection). These algorithms are more practical for real life usage as they don't require a labeled dataset. However, they have higher false positive rates and are more difficult to evaluate. Another research direction that may enable real life usage is to evaluate our approach on on-line clustering and classification algorithms.

2. **Clustering algorithms improvement.** Each of the CLB algorithms has its own drawbacks, which can be further mitigated. For instance, (a) our proposed WK-means yields clusters that are not balanced enough for a load balancing application. Further research can be conducted in order to improve its load balance. (b) For K-means, the process of selecting features subsets was done manually. Further research may find better ways of selecting optimal features (that yield the most balanced clusters). (c) The runtime of XLK-means and Multipart clustering algorithms can be improved using native and parallel implementation. XLK-means may converge faster using different initiation scheme. Multipart may run faster using better Pythonic interface to `METIS` and using less neighbors.

3. **Further evaluation.** We evaluate the approach on a single dataset. Although we use common practices to avoid over-fitting, such as, split to train and test sets, and cross-validation it would be interesting to validate our approach on additional traces. In addition, it would be interesting to research whether the approach can be generalized to other ML-based appliances that require scaling.

Appendix A

K-means weighting scheme

K-means is well defined only for Euclidean spaces, where distance between vectors A and B is expressed as $||A - B|| = \sqrt{\sum_i (A_i - B_i)^2}$. Let W be a weights vector, w_i denotes the weight of the i-th feature. The weighted distance can be expressed as

$$\begin{aligned} ||A - B||_W &= \sqrt{\sum_i w_i (A_i - B_i)^2} = \sqrt{\sum_i (\sqrt{w_i}(A_i - B_i))^2} \\ &= \sqrt{\sum_i (\sqrt{w_i} A_i - \sqrt{w_i} B_i)^2} = ||\sqrt{W} A - \sqrt{W} B|| \end{aligned}$$

Thus, in order to give a weight of w_i to the i-th feature, the i-th element of all vectors should be multiplied by $\sqrt{w_i}$.

It is a common practice to standardize data for K-means such that every feature has zero mean and unit variance. The unit variance is achieved by dividing every feature by its standard deviation. According to the weighting scheme it is the equivalent of giving weight of $1/\sigma^2$ (the variance).

Bibliography

[AB00] Luis Aversa and Azer Bestavros. 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*, pages 24–29. IEEE, 2000.

[AFAA08] Adebayo O Adetunmbi, Samuel O Falaki, Olumide S Adewale, and Boniface K Alese. Network intrusion detection based on rough set and k-nearest neighbour. *International Journal of Computing and ICT Research*, 2(1):60–66, 2008.

[BBD00] PS Bradley, KP Bennett, and Ayhan Demiriz. Constrained k-means clustering. *Microsoft Research, Redmond*, pages 1–8, 2000.

[BCLM98] Azer Bestavros, Mark Crovella, Jun Liu, and David Martin. Distributed packet rewriting and its application to scalable server architectures. In *Network Protocols, Proceedings. Sixth International Conference on*, pages 290–297. IEEE, 1998.

[BG06] Arindam Banerjee and Joydeep Ghosh. Scalable clustering algorithms with balancing constraints. *Data Mining and Knowledge Discovery*, 13(3):365–395, 2006.

[BMS$^+$16] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In *Algorithm Engineering*, pages 117–158. Springer, 2016.

[CNMY14] Xiaojun Chang, Feiping Nie, Zhigang Ma, and Yi Yang. Balanced k-means and min-cut clustering. *CoRR*, abs/1411.6235, 2014.

[F515] F5. Deploying the BIG-IP LTM with Multiple BIG-IP AAM and ASM Devices, 2015.

[FBIM17] Gal Frishman, Yaniv Ben-Itzhak, and Oded Margalit. Cluster-based load balancing for better network security. In *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, pages 7–12. ACM, 2017.

[FM88] Charles M Fiduccia and Robert M Mattheyses. A linear-time heuristic for improving network partitions. In *Papers on Twenty-five years of electronic design automation*, pages 241–247. ACM, 1988.

[GCC14] Nuwan Ganganath, Chi-Tsun Cheng, and K Tse Chi. Data clustering with cluster size constraints using a modified k-means algorithm. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2014 International Conference on*, pages 158–161. IEEE, 2014.

[GdB12] Guilherme Galante and Luis Carlos E de Bona. A survey on cloud computing elasticity. In *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, pages 263–270. IEEE, 2012.

[HGW$^+$14] Weiming Hu, Jun Gao, Yanguo Wang, Ou Wu, and Stephen Maybank. Online adaboost-based parameterized methods for dynamic distributed network intrusion detection. *IEEE Transactions on Cybernetics*, 44(1):66–82, 2014.

[HL95] Bruce Hendrickson and Robert Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing*, 16(2):452–469, 1995.

[HSC$^+$11] Shi-Jinn Horng, Ming-Yang Su, Yuan-Hsin Chen, Tzong-Wann Kao, Rong-Jian Chen, Jui-Lin Lai, and Citra Dwi Perkasa. A novel intrusion detection system based on hierarchical clustering and support vector machines. *Expert systems with Applications*, 38(1):306–313, 2011.

[HXSZ09] Ruhan He, Weibin Xu, Jiaxia Sun, and Bingqiao Zu. Balanced k-means algorithm for partitioning areas in large-scale vehicle routing problem. In *Intelligent Information Technology Application, 2009. IITA 2009. Third International Symposium on*, volume 3, pages 87–90. IEEE, 2009.

[JSD05] Wenbao Jiang, Hua Song, and Yiqi Dai. Real-time intrusion detection for high-speed networks. *Computers & security*, 24(4):287–294, 2005.

[KBM94] Eric Dean Katz, Michelle Butler, and Robert McGrath. A scalable http server: The ncsa prototype. *Computer Networks and ISDN systems*, 27(2):155–164, 1994.

[KK98] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[KL70] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[KLK14] Gisung Kim, Seungmin Lee, and Sehun Kim. A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. *Expert Systems with Applications*, 41(4):1690–1700, 2014.

[KMS12] Levent Koc, Thomas A Mazzuchi, and Shahram Sarkani. A network intrusion detection system based on a hidden naïve bayes multiclass classifier. *Expert Systems with Applications*, 39(18):13492–13500, 2012.

[LASB08] Anh Le, Ehab Al-Shaer, and Raouf Boutaba. On optimizing load balancing of intrusion detection and prevention systems. In *INFOCOM Workshops 2008, IEEE*, pages 1–6. IEEE, 2008.

[LHNL17] Hanyang Liu, Junwei Han, Feiping Nie, and Xuelong Li. Balanced clustering with least square regression. In *AAAI*, pages 2231–2237, 2017.

[LKT15] Wei-Chao Lin, Shih-Wen Ke, and Chih-Fong Tsai. Cann: An intrusion detection system based on combining cluster centers and nearest neighbors. *Knowledge-based systems*, 78:13–21, 2015.

[LXZ$^+$12] Yinhui Li, Jingbo Xia, Silan Zhang, Jiakai Yan, Xiaochuan Ai, and Kuobin Dai. An efficient intrusion detection system based on support vector machines and gradually feature removal method. *Expert Systems with Applications*, 39(1):424–430, 2012.

[MF14] Mikko I Malinen and Pasi Fränti. Balanced k-means for clustering. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 32–41. Springer, 2014.

[Mog95] Jeffrey C Mogul. Network behavior of a busy web server and its clients. 1995.

[PP07] Mrutyunjaya Panda and Manas Ranjan Patra. Network intrusion detection using naïve bayes. *International journal of computer science and network security*, 7(12):258–263, 2007.

[PVG$^+$11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

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

[SWF05] Lambert Schaelicke, Kyle Wheeler, and Curt Freeland. Spanids: a scalable network intrusion detection loadbalancer. In *Proceedings of the 2nd Conference on Computing Frontiers*, pages 315–322. ACM, 2005.

[Tar06] Willy Tarreau. Haproxy-the reliable, high-performance tcp/http load balancer, April 2006.

[TBLG09] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–6. IEEE, 2009.

[TLX09] Daxin Tian, Yanheng Liu, and Yang Xiang. Large-scale network intrusion detection based on distributed learning algorithm. *International Journal of Information Security*, 8(1):25–35, 2009.

[VSL+07] Matthias Vallentin, Robin Sommer, Jason Lee, Craig Leres, Vern Paxson, and Brian Tierney. The nids cluster: Scalable, stateful network intrusion detection on commodity hardware. In *International Workshop on Recent Advances in Intrusion Detection*, pages 107–126. Springer, 2007.

[Wat17] Ken Watford. METIS for Python. `https://github.com/kw/metis-python`, 2012–2017. [Online; accessed 10-Oct-2017].

[XCA+06] Konstantinos Xinidis, Ioannis Charitakis, Spyros Antonatos, Kostas G Anagnostakis, and Evangelos P Markatos. An active splitter architecture for intrusion detection and prevention. *IEEE Transactions on Dependable and Secure Computing*, 3(1):31–44, 2006.

[YMC11] Sun Yuepeng, Liu Min, and Wu Cheng. A modified k-means algorithm for clustering problem with balancing constraints. In *Measuring Technology and Mechatronics Automation (ICMTMA), 2011 Third International Conference on*, volume 1, pages 127–130. IEEE, 2011.

[Yu02] Philip Shi-lung Yu. Loading balancing across servers in a computer network, February 26 2002. US Patent 6,351,775.

[ZMP11] Jilian Zhang, Kyriakos Mouratidis, and Hwee Hwa Pang. Heuristic algorithms for balanced multi-way number partitioning. 2011.

[ZWL10] Shunzhi Zhu, Dingding Wang, and Tao Li. Data clustering with size constraints. *Knowledge-Based Systems*, 23(8):883–889, 2010.

[ZZH08] Jiong Zhang, Mohammad Zulkernine, and Anwar Haque. Random-forests-based network intrusion detection systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(5):649–659, 2008.

תקציר

בעידן נתוני עתק, כמות התעבורה הולכת וגדלה. לכן, נעשה שימוש רב בשיטות גידול (סילומיות). למשל, שירות רשתי הבנוי ממספר מופעים של אותו רכיב יחד עם מאזן עומסים שמחלק את התעבורה הנכנסת בין המופעים.

בעוד השיטה הנפוצה ביותר לאיזון עומסים מבוססת על "ראונד־רובין", ישנן גישות לאיזון מיטבי הלוקחות בחשבון גם את פונקציונליות השירות. למשל, איזון עומסים של שרתי "פרוקסי" המגדיל את יחס ההמצאות בזכרון המטמון.

במאמר זה, אנו מציגים גישות חדשות לאיזון עומסים עבור מערכות אבטחת רשתות מחשבים מבוססות למידה. הגישות המוצעות עושות שימוש בשיטות אשכול (CLUSTERING) תוך שמירה על עומס מאוזן בין רכיבי המערכת. אנו מדגימים כי גישות אלו הן גם ברות גידול וגם משפרות את ביצועי המערכת הלומדת, בהשוואה לגישות איזון עומסים מסורתיות.

לתרומה של עבודה זו שלושה חלקים. ראשית, אנו מציגים סקר עדכני של אלגוריתמי אשכול מכוונים איזון ודנים בהתאמתם לבעיית איזון עומסי תקשורת. שנית, אנו מעריכים ומדגימים את השיפור בביצועי מערכות אבטחת רשתות מחשבים לומדות כתוצאה משימוש בגישות שלנו בהשוואה אל מול גישות איזון עומסים מסורתיות. שלישית, אנו משחררים את קוד המקור של מערכת ההערכה שפיתחנו ואת המימוש שלנו לאלגוריתמי האשכול השונים במטרה לאפשר שחזור של התוצאות שלנו ומחקר עתידי.

עבודה זו בוצעה בהדרכתו של דר' יניב בן־יצחק ממעבדת המחקר של IBM ובליוויה של פרופ' ענת ברמלר־בר מבי"ס אפי ארזי למדעי המחשב, המרכז הבינתחומי, הרצליה.

חלק מן התוצאות בחיבור זה פורסמו כמאמרים מאת המחבר ושותפיו למחקר בכנסים ובכתבי־עת במהלך תקופת המחקר של המחבר, אשר גרסאותיהם העדכניות ביותר הינן:

Gal Frishman et al. Cluster-based load balancing for better network security. In *Proceedings of Big-DAMA '17*. ACM, 2017.

תודות

ברצוני להודות למנחה שלי, ד"ר יניב בן־יצחק על ההדרכה והעידוד שנתן לי במהלך התיזה. תודות לו, זכיתי לחויה מדהימה בהצגת העבודה בכנס Big-DAMA '17.

כמו כן, ברצוני להודות לד"ר עודד מרגלית על הקריאה הביקורתית של המאמר ועל תובנות המחקר שנתן לי.

לסיום, ברצוני להביע הכרת תודה לאשתי, אריאלה, על תמיכתה המתמשכת ועל שעודדה אותי לסיים את המחקר והמאמר ואפשרה לי את הזמן והשקט הנפשי לבצע זאת.

המרכז הבינתחומי בהרצליה

בית־ספר אפי ארזי למדעי המחשב

התכנית לתואר שני (M.Sc.) - מסלול מחקרי

איזון עומסים מבוסס אשכול לשיפור אבטחת רשתות מחשבים

מאת

גל פרישמן

עבודת תזה המוגשת כחלק מהדרישות לשם קבלת תואר מוסמך M.Sc.
במסלול המחקרי בבית ספר אפי ארזי למדעי המחשב, המרכז הבינתחומי הרצליה

ינואר 2018