Evaluating Machine Learning Algorithms for Anomaly Detection in Clouds

Anton Gulenko, Marcel Wallschläger, Florian Schmidt, Odej Kao
Complex and Distributed IT-Systems
TU Berlin
Berlin, Germany
{*firstname*}.{*lastname*}@*tu-berlin.de*

Feng Liu
European Research Center
Huawei
Munich, Germany
feng.liu1@huawei.com

***Abstract*—Critical services in the field of Network Function Virtualization require elaborate reliability and high availability mechanisms to meet the high service quality requirements. Traditional monitoring systems detect overload situations and outages in order to automatically scale out services or mask faults. However, faults are often preceded by anomalies and subtle misbehaviors of the services, which are overlooked when detecting only outages.**

We propose to exploit machine learning techniques to detect abnormal behavior of services and hosts by analysing metrics collected from all layers and components of the cloud infrastructure. Various algorithms are able to compute models of a hosts normal behavior that can be used for anomaly detection at runtime. An offline evaluation of data collected from anomaly injection experiments shows that the models are able to achieve very high precision and recall values.

***Keywords*-anomaly detection; machine learning; cloud; network function virtualization**

I. Introduction

Many services rely on a cloud infrastructure to provide the elasticity needed for a scalable deployment and short time-to-market cycles. A side effect of Infrastructure as a Service (IaaS) deployments is a rather deep technology stack. This is exacerbated in Network Function Virtualization (NFV) scenarios, where critical services have especially high demands for service quality. An availability of up to 99,9999% is often expected from telecommunication service providers, which is hard to achieve with open source cloud software and commodity hardware.

Traditional reliability approaches monitor system components for outages, using active or passive monitoring techniques like heartbeats or status request. These mechanisms only detect faults after they have occurred. However, faults are often preceded by abnormal system behavior or degraded system states, that can only be observed in fine-grained monitoring data obtained inside the affected host. Detecting and handling such anomalies can avoid faults before they happen and generally increase system throughput.

Machine learning algorithms provide the means to learn from historical observations and use this knowledge to get predictions about the future. Applied to anomaly detection, this means to first observe a host both while it experiences anomalies and during normal operation. A machine learning algorithm is then used to convert the observed data to a model representing the hosts normal and abnormal behavior. The resulting model can now make estimations about whether the host is acting normally by observing it at runtime.

This paper presents an evaluation of 13 offline classification algorithms used for detecting anomalies on a set of hosts. The evaluation is based on a cloud testbed running a scenario of Virtualized Network Function (VNF) services. The data for training and evaluating the machine learning algorithms is obtained through controlled anomaly injection experiments and deep, cross-layer data monitoring. A ten-fold cross-validation shows that some algorithms achieve an exceptionally high average F1-measure. These results suggest that the monitoring data indeed contains insights about abnormal host behavior, which makes the concept of using machine learning for anomaly detection applicable to productive systems.

The remainder of this paper is organized as follows. Section II outlines related efforts in the field of anomaly detection in clouds. Section III describes the concept of using classification algorithms for anomaly detection. Section IV describes the testbed setup, anomaly injection and data collection techniques used to obtain the data for the evaluation. Section V describes the evaluation procedure and presents the evaluation results. Finally, section VI summarizes the results and concludes.

II. Related Work

Anomaly detection is a well-established research field with many theoretical and practical application areas. In this section we focus on more recent work targeting anomaly detection specifically in the cloud computing and NFV domain.

The work of Miyazawa et al. follows attempts use machine learning techniques to detect anomalies in cloud environments [1]. The prototype system is called "vNMF" and uses Self Organizing Maps (SOM) for distinguishing anomalies from normal behavior. The "vNMF" system targets NFV environments while our approach is generically applicable to any scenario involving virtualized services.

978-1-4673-9005-7/16/$31.00 ©2016 IEEE

SOM is an unsupervised learning algorithm based on neural networks. It is capable of analysing different types of data while having the advantage of using little amount of computer resources. SOM maps vectorial input data with similar patterns into the same locations so neighboring locations represent similar data patterns [2]. Our evaluation is not focused on a single algorithm, but compares a variety of different machine learning algorithms. The ones with the best precision and recall results have the biggest potential to be used in practice.

The "vNMF" system relies on a decentralized approach: data analysis components are running on the same machines as the VNFs. Our approach follows a similar architectural design, including a monitoring, data analysis and notification component. "vNMF" includes an additional database for persistent storage of collected metrics. Regarding the collected metrics, our approach collects a wide range of hypervisor specific metrics as well as process specific information such as resource usage per process. The additional information can potentially help detect anomalies that would remain hidden otherwise.

Miyazawa et al. evaluated their system on a setup of two physical machines executing three virtual machines each. Three failure scenarios were used to evaluate the "vNMF" prototype running on the machines. The first use case is a memory leak inside a database running in a virtual machine. The allocated memory peaks after approximately two hours, resulting in high swap activity, which also influences the CPU and disk I/O of the server. For the second use case the memory leak is combined with a simulated VNF, generating network traffic between two Virtual Machines (VMs). The third scenario consists of bi-directional 80Mbps traffic between two virtual machines. For the evaluation Miyazawa et al. trained their model with data collected from idle machines. The evaluation presented in this paper includes a wider range of anomaly scenarios and a larger, more realistic testbed.

Another approach published by Sauvanaud et al. also targets anomaly detection in NFV scenarios [3]. Similar to our approach, Project Clearwater is used as a virtualized IP-Multimedia Subsystem (vIMS) and the prototype is evaluated with fault scenarios like memory leak, high cpu usage and network related faults such as high latency and packet loss. The Random Forest algorithm is used as a supervised learning method to detect anomalies based on monitoring data.

III. Classification for Anomaly Detection

Supervised classification algorithms like decision trees or support vector machines (SVM) often have a common interface for training and testing models. The following defines this interface in the context of host-based anomaly detection. Training is done through a function $train(S_1 \dots S_n)$ where each $S_i \in \mathbb{R}^m \times \mathbb{L}$ is a labeled sample of m metrics taken from the observed host. The set $\mathbb{L}$ is the set of all possible labels. In the context of anomaly detection we choose $\mathbb{L} = \{normal, abnormal\}$. The $train$ function computes a classification model in the form of a function $classify : \mathbb{R}^m \to \mathbb{L}$. In other words, the $classify$ function receives an unlabeled sample and computes a predicted label from $\mathbb{L}$.

Training is typically done in a separate offline phase and can be computationally expensive, while the $classify$ function evaluates quickly enough to be used in real-time. In order to perform the training, a sufficient number of S_i samples must be recorded in advance. Samples with the label $normal$ can be easily recorded while the host is executing normally. This requires a human administrator to manually assert normal operation of the host. Obtaining $abnormal$ samples is more challenging, as they require artificial injection of anomalies into the host. A framework for controlled anomaly injection must be implemented and executed on the host while labelling all observed samples as $abnormal$. For both $normal$ and $abnormal$ samples, it is important to put the host in different load situations to capture all behavior patterns that can be expected later, during runtime. This requires a second framework for generating realistic workloads that change throughout the course of the data collection.

The following subsections give short overviews over the machine learning algorithms that are included in the evaluation. A prototypical implementation of above frameworks and data collection procedures is used to obtain the necessary training and test data.

1) J48: J48 is an implementation of the C4.5 decision tree learning algorithm by Quinlan [4]. The C4.5 algorithm builds the decision tree by splitting up nodes on the metric with the highest normalized information gain. Starting from one node, the tree is split up until the information gain falls below a predefined threshold. In an optional pruning phase, the resulting tree is pruned from the leaf nodes upwards to decrease overfitting effects.

2) Logistic Model Tree: LMT [5] uses the C4.5 decision tree splitting criterion, but works with logistic regression models in the tree nodes, instead of the constant values used in simple decision trees. The LogitBoost algorithm [6] is used to efficiently compute the regression models. A test sample is classified by traversing the tree from the root node and testing the input sample against each contained regression model to decide which path should be followed down the tree.

3) Hoeffding Tree: This online decision tree learning algorithm was proposed by Hulten et al. [7]. As the online scenario does not define bounds for the number of samples, the authors use the Hoeffding bound [8] to predict the number of samples in order to approximate the deviation of the unbounded data stream. Taking the number of samples into account, the Hoeffding Tree algorithm operates on a

windows of data. The splitting criterion is the information gain, as used in the C4.5 algorithm.

4) Rep Tree: The Reduced Error Pruning Tree applies additional reduced-error pruning as a post-processing step of the C4.5 algorithm.

5) Random Tree: The Random Tree classifier constructs a decision tree by randomly choosing the features evaluated for each node [9].

6) Decision Stump: The decision stump classification algorithm is based on the C4.5 decision tree learning algorithm, but uses only one layer for decisions [10]. The algorithm stops learning when the best feature with the most information gain is added.

7) Decision Table: This class of classifier algorithms maps rules to learned classes [11]. In general, decision tables divide up the large hypothesis space to smaller areas. In machine learning, an optimal feature sub set is chosen for a decision table representation.

8) Random Forests: The Random Forests classifier [12] uses a multitude of decision trees, each trained with a random sub set of the total available features.

9) JRIP: JRip is a classification algorithm learning a set of rules and was proposed by Cohen [13]. JRip uses incremental pruning to reduce the error by finding core rules describing a class to be learned.

10) ONER: ONER [14] is a rule based classification algorithm using minimum-error attributes for prediction.

11) PART: Frank and Witten [15] introduced the PART algorithm, which uses lists of decision rules as prediction model. For each class a C4.5 decision tree is created and the best leaf is chosen to represent a rule for the class.

12) Naive Bayes: Naive Bayes [16] learns a Bayesian network as an underlying model. A Bayesian network consists of a graphical model indicating dependencies between values with associated conditional probabilistic tables. Naive Bayes uses a strong conditional independence assumption, which states that all attributes of a sample are independent, in order to efficiently learn the model.

13) SMO: Sequential Minimal Optimization [17] is an optimized version of Support Vector Machines (SVMs). As SVMs need to solve large quadratic programming problems, SMO divides it into smaller quadratic programming sub-problems in order to efficiently learn large sets of data.

IV. Experimental Setup

The training and test data used for the evaluation of above classification algorithms is collected in a dedicated testbed. This section describes the testbed environment, including the components for collecting data and injecting anomalies in monitored hosts.

The evaluation testbed is based on 13 physical machines, each equipped with an Intel Xeon X3450 (4 cores), 16GB Ram, three 1TB disk and two 1GBit Ethernet interfaces. The cloud environment is based on a high available OpenStack installation [18]. Since a number of hosts are occupied by the redundant OpenStack services, only a sub set of 6 physical hosts are included in the evaluation. The KVM is used as virtualization engine, while Open vSwitch is provides virtual networking capabilities.

Figure 1 shows the testbed infrastructure including all hosts that are monitored by the data collection agent. The data collection and anomaly injection is performed on a set of 6 physical hosts and 10 virtual machines. OpenStacks own orchestration system Heat is used alongside with Ansible to manage the VNF services.

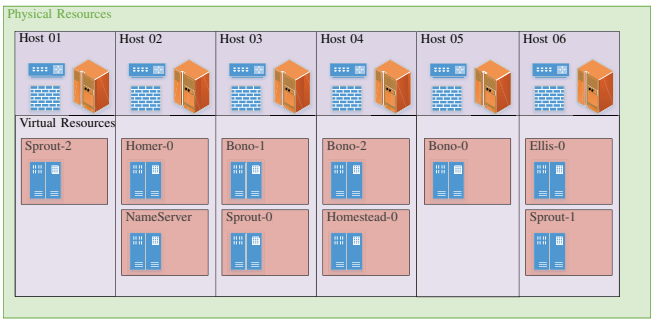

Figure 1. Testbed setup

The services of Project Clearwater used as example VNFs. Project Clearwater is an open source vIMS implementation. The following gives an overview over the components of Project Clearwater.

Bono SIP edge proxy implementing the Proxy Call Session Control Function.

Sprout SIP registrar and message router, handles client authentication and the ISC interface to application servers. Implements the I-CSCF and S-CSCF functions.

Homestead Repository of subscriber credentials and service profiles.

Homer Repository of subscriber-controlled documents for each user.

Ellis Sample provisioning portal providing subscribers to be configured in Homestead and Homer.

Sip-stress SIP endpoint emulation executing defined call scenarios.

DNS Name service to enable load balancing for multi-instance services.

The Bono and Sprout services are running in a replicated configuration. The Sip-stress service executes a series of IP Multimedia Subsystem (IMS) commands and messages described in an XML format. The executed scenario consists of two phases: a registration phase and a call phase. The simulated IMS clients first register at the Project Clearwater services and then start generating calls, while maintaining a controlled load level in the system.

A. Data Collection

The data collection service samples and provides various metrics in the form of time series data. While running locally on every host, it can store the collected data to log files or send them over the network for the purpose of visualization or further analysis. The frequency of sampling data can be arbitrarily high and the data collection service is designed to minimize resource overhead induced by the sampling routine itself. While collecting data for the evaluation presented in section V the data is sampled in 250 millisecond intervals. The Linux kernel provides resource usage information through the `/proc` filesystem, which is the main data source for the data collection service.

A hypervisor host provides additional interfaces for sampling data about hosted virtualized resources. The data collection service implements data collection from the virtualization Application Programming Interface (API) Libvirt [19] and the OVSDB [20] protocol provided by the software switch Open vSwitch [21]. These APIs provide additional metrics about virtual machines and virtual network interfaces. Table I gives an overview over the metrics available from the different data sources.

B. Anomaly Injection

Anomaly injections are used to generate labelled training and testing data. Therefore, a variety of anomaly scenarios are injected into the monitored hosts. The anomaly injection experiments are executed in a loop. Each anomaly is injected for a random duration between 5 and 15 minutes, followed by a cooldown phase of 30 to 60 seconds. After injecting a given anomaly on every relevant host, the next anomaly is chosen to continue the experiments. After each anomaly injection, the system load, defined by the amount of registrations and calls per second, are randomly chosen and reconfigured. The implemented anomalies are listed and described in the following. Every anomaly has custom parameters, which are randomly chosen within a predefined range.

1) Memory Leak: A process with constantly growing memory usage is started on one of the VMs or hypervisor machine. The memory leak causes a growing memory usage of 20 to 30 MB on a VM and 400 to 600 MB on a hypervisor each 1 to 2 seconds. This fast growing memory demand quickly leads to high swapping and CPU activity.

2) Excessive CPU Usage: This scenario injects a process with a constantly high CPU usage. It uses the tool `stress-ng`[1] to run between 2 and 4 workers spinning on mathematical operations.

3) Disk Write: The Disk Write scenario uses the hdd function of `stress-ng` for spinning on write()/unlink() filesystem operations, spawning between 1 and 4 workers

4) Packet Loss: This network level scenario defines rules via the `tc` tool (short for "traffic control"), which continuously drops between 5% and 20% of all packets going through all network interfaces of the injected host.

5) Increased Latency: Another network level scenario, which uses iptables[2] to increase the latency for each incoming and outgoing packet by 70 to 250 milliseconds.

6) Throttle Bandwidth: This scenario reduces the maximal throughput of one network device of the injected host to between 100 and 2000 Kilobyte using the `tc` tool.

7) Bandwidth Usage: A long lasting download is executed on the injected host, which tries to occupy as much network bandwidth as possible. The downloaded data is not stored on the hard drive to limit the impact of this scenario to the network.

V. Evaluation Results

The following evaluations are based on the data collected on a set of two physical and two virtual hosts. The data was collected over a period of 72 hours, including equal time periods for normal operation and anomaly injection. The result tables contain the average evaluation results for a subset of four hosts, which have been computed independently. The set of observed hosts consists of two physical and two virtual machines running the Project Clearwater services Bono and Sprout.

A. Cross Validation

This section presents the results of a ten-fold cross validation performed on the collected data. Two data sets form the input for the validation: the normal operation data and the anomaly data obtained through anomaly injection. Both data sets are shuffled and split into twenty equally sized sub sets, ten from each original data set. Randomly chosen pairs of normal operation sub sets and anomaly sub sets then form the ten validation data sets.

The evaluation uses the following procedure to process every combination of validation data set and evaluated algorithm. First, model is trained based on the selected validation data set. Then, the remaining 9 data sets are usedq for evaluating the resulting model by computing the confusion matrix, the precision, recall and F1 score. The F_1 score is the harmonic mean of the precision and the recall and is computed with the following formula:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Table II summarizes the average precision, recall and F_1 score for every evaluated algorithm. The algorithms are sorted in a descending manner, based on their F_1 score.

The results show that multiple algorithms are able to predict anomalies with an exceptionally high F_1 measure of above 98%. The Hoeffding Tree algorithm is an exception

[1] http://kernel.ubuntu.com/~cking/stress-ng

[2] https://www.netfilter.org/projects/iptables/index.html

Metric	Description	System-wide	Per process	Libvirt: Per VM	OVSDB: Per bridge
CPU	CPU utilization	×	×	×	
RAM	RAM utilization	×	×	×	
Disk IO	Hard disk access in bytes and access times	×	×	×	
Disk space usage	Disk space per partition	×		×	
Network IO	Network utilization in bytes and packets	×	×	×	×
Network protocol counters	Protocol specific counters for IP, UDP, TCP and ICMP	×			
Number of processes		×			
Number of threads			×		

Table I
SYSTEM RESOURCES SAMPLED BY THE DATA COLLECTION SERVICE

Algorithm	Precision	Recall	F_1
Random Forest	99.35%	99.28%	99.31%
LMT	99.01%	98.98%	98.99%
J48	98.95%	98.92%	98.93%
JRIP	98.88%	98.80%	98.84%
PART	98.77%	98.66%	98.72%
Rep Tree	98.72%	98.69%	98.71%
Random Tree	98.29%	98.28%	98.29%
Decision Table	96.27%	96.28%	96.27%
ONER	95.30%	95.27%	95.28%
SMO	93.74%	88.66%	91.13%
Decision Stump	89.20%	88.27%	88.74%
Naive Bayes	90.16%	85.15%	87.59%
Hoeffding Tree	45.73%	50.04%	47.79%
Average	92.49%	91.95%	92.20%

Table II
TEN-FOLD CROSS-VALIDATION RESULTS

with an F_1 of only 47.79%. This is likely due to the fact that the Hoeffding Tree approach allows to build the model from streaming data, which means that a lot less information is available compared to analysing the entire training set at once. The streaming capabilities of this algorithm are not exploited in the given evaluation.

B. Time-based Evaluation

Models created from monitoring data collected from hosts are affected by aging effects. The models are created once from historical data, and then used for a certain time frame, until they are possibly re-trained. To evaluate this aging aspect of host monitoring data, a second evaluation is conducted.

The time-based evaluation does not shuffle the test and training data sets, but instead keeps them in the order they have been recorded in the monitored hosts. The entire timeline of all samples is separated in 9 equally sized sub sets. The first three sub sets are used to train three different models for every evaluated algorithm. The resulting models are then evaluated using the last six sub sets.

Table III summarizes the results of the time-based evaluation. The average F_1 score drops to 69%, compared to the cross-validation average of 92%. This suggests that the created models indeed suffer from aging effects: over time, the metrics collected on hosts tend to drift in different directions, e.g. disk usage, or make sudden jumps, e.g. memory freed from ended processes. The time difference between the training and testing data was between 2 and 3 days.

Interesting further tests would include data collected over longer periods of time to investigate how the aging effects behave. Ways to counter the aging effect include continuously updating the models or periodically creating new models. The first approach requires algorithms to be capable of online updates of models, while the latter has the drawback of periodically consuming compute resources, making this approach not usable in all productive scenarios.

VI. CONCLUSION

This paper evaluates the usage of a selection of machine learning algorithms to detect anomalies in cloud infrastructures. For collecting the evaluation data sets, an architecture for collecting data and injecting anomalies was presented. The two frameworks were used to collected data from normal operation and anomaly injection experiments over the course of 72 hours.

The results of a ten-fold cross-validation suggest that machine learning algorithms are able to predict anomalies with both high precision and high recall measures, the average F_1 score being 92%. However, an additional evaluation that separated the training and test data sets by time resulted in a lower F_1 measure of 69%.

The second evaluation shows that host-based behavior models suffer from aging effects that must be countered

Algorithm	Precision	Recall	F_1
Random Forest	75.83%	71.77%	73.65%
LMT	81.48%	78.67%	79.98%
J48	81.86%	78.65%	80.12%
JRIP	68.28%	62.43%	65.13%
PART	83.56%	80.10%	81.70%
Rep Tree	74.02%	70.40%	72.08%
Random Tree	63.54%	61.38%	62.41%
Decision Table	66.12%	63.97%	65.00%
ONER	64.75%	61.94%	63.25%
SMO	79.92%	78.52%	79.20%
Decision Stump	74.82%	72.09%	73.41%
Naive Bayes	73.98%	69.08%	71.20%
Hoeffding Tree	30.59%	50.00%	35.65%
Average	70.67%	69.15%	69.45%

Table III
TIME-BASED EVALUATION RESULTS

when using the given approach in production. Ideas for countering the aging effect include periodical re-training or continuous updates of the models. Measuring the exact impact of the aging is an interesting future effort, along with finding solutions to the problem that can be used in productive systems.

REFERENCES

[1] M. Miyazawa, M. Hayashi, and R. Stadler, "vnmf: Distributed fault detection using clustering approach for network function virtualization," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 640–645.

[2] T. Kohonen, M. R. Schroeder, and T. S. Huang, Eds., *Self-Organizing Maps*, 3rd ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001.

[3] C. Sauvanaud, K. Lazri, K. Kanoun *et al.*, "Towards black-box anomaly detection in virtual network functions," in *International Conference on Dependable Systems and Networks Workshop*. IEEE, 2016, pp. 254–257.

[4] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[5] N. Landwehr, M. Hall, and E. Frank, "Logistic model trees," *Machine Learning*, vol. 59, no. 1-2, pp. 161–205, 2005.

[6] J. Friedman, T. Hastie, R. Tibshirani *et al.*, "Additive logistic regression: a statistical view of boosting," *The annals of statistics*, vol. 28, no. 2, pp. 337–407, 2000.

[7] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2001, pp. 97–106.

[8] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American statistical association*, vol. 58, no. 301, pp. 13–30, 1963.

[9] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[10] W. Iba and P. Langley, "Induction of one-level decision trees," in *Proceedings of the ninth international conference on machine learning*, 1992, pp. 233–240.

[11] R. Kohavi, "The power of decision tables," in *8th European Conference on Machine Learning*. Springer, 1995, pp. 174–189.

[12] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[13] W. W. Cohen, "Fast effective rule induction," in *Proceedings of the twelfth international conference on machine learning*, 1995, pp. 115–123.

[14] R. C. Holte, "Very simple classification rules perform well on most commonly used datasets," *Machine learning*, vol. 11, no. 1, pp. 63–90, 1993.

[15] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization," 1998.

[16] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1995, pp. 338–345.

[17] J. C. Platt, "12 fast training of support vector machines using sequential minimal optimization," *Advances in kernel methods*, pp. 185–208, 1999.

[18] S. H. Makhsous, A. Gulenko, O. Kao, and F. Liu, "High available deployment of cloud-based virtualized network functions," in *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE, 2016, pp. 468–475.

[19] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehörster, and A. Brinkmann, "Non-intrusive virtualization management using libvirt," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 574–579.

[20] B. Pfaff and B. Davie, "The open vswitch database management protocol," 2013. [Online]. Available: https://tools.ietf.org/html/rfc7047

[21] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *12th USENIX symposium on networked systems design and implementation (NSDI 15)*, 2015, pp. 117–130.