Root Cause Analysis of Noisy Neighbors in a Virtualized Infrastructure

Hedi Bouattour, Yosra Benslimen, Marouane Mechteri, Hanane Biallach

Root Cause Analysis of Noisy Neighbors in a Virtualized Infrastructure

Hedi Bouattour
Orange Labs
Chatillon, France
hedibouattour2010@gmail.com

Yosra Ben Slimen
Orange Labs
Chatillon, France
yosra1.benslimen@orange.com

Marouane Mechteri
Orange Labs
Chatillon, France
mechtri.marwen@gmail.com

Hanane Biallach
Orange Labs
Chatillon, France
hanane.biallach@orange.com

***Abstract*—This paper proposes a model to identify the noise source in a virtualized infrastructure. This phenomenon appears when network functions running under virtual machines that are deployed on the same physical server compete for physical resources. First, an anomaly detection model is proposed to identify the machines that are in an abnormal state in the infrastructure by performing an unsupervised learning. An investigation of the root cause is later achieved by searching how anomalies are propagated in the system. To do this, a supervised learning of the anomaly propagation paths is proposed. A propagation graph is automatically created with a score assigned to its components. With a testbed created using Openstack, an experimentation study with real data is held giving promising results.**

***Index Terms*—Virtualization, mobile networks, anomaly detection, root cause analysis, noisy neighbor, machine learning**

I. Introduction

5G subscriptions are predicted to reach 1.9 billion by 2025 where 35% of traffic will be carried by 5G networks [1]. This means that 65% of the global population are predicted to be covered by the technology. In order to have an increased speed, performance, scalability, and flexible service deployment, 5G technology relies on Network Function Virtualization (NFV, [2]). This latter is known as the act of virtualizing the physical network functions in order to provide dynamic/on-demand communication services. NFV is the deployment of one or more virtual machines running different software and processes, on top of standard high-volume servers, switches and storage devices, or even cloud computing infrastructure as illustrated in Fig. 1.

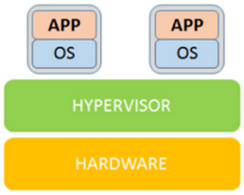

Fig. 1. Virtual machines.

The advantage of NFV is to deliver high-performance networks with greater scalability, elasticity, and adaptability at reduced costs compared to networks built from traditional equipment. However, this technology may arise new challenges such as the extreme usage of shared resources which may result in a noisy neighbor situation. This phenomenon occurs when applications or virtual machines running on the same hypervisor compete for resources, resulting in a degradation of performance [3]–[6].

In literature, some research works propose preventive approaches that aim to mitigate the noisy neighbors' effects on shared processor resources such as in [5]. Whereas several research works aim at detecting the noisy neighbors at real time. Some works rely on a static analysis of traffic patterns in order to propose thresholds-based methods such as in [7] while other works propose machine learning solutions. In [3], the noisy neighbors are detected with support vector machines and random forests [8] with an accuracy that is higher than 90%. Authors in [4] propose exploiting time-series data and applying a convolutional neural network [8] for noisy neighbors detection.

Once the noisy neighbor detected, one possible solution is to migrate the affected machines to other servers except that the target server may suffer itself from degraded performance as well. Hence, the detection phase alone is not sufficient since the source of the problem remains unknown, especially when it comes to very high dimensions. Performing a root cause analysis (RCA) is of a paramount importance.

RCA is widely used in virtual infrastructures for several types of anomalies by using Bayesian networks [9] or propagation graphs that describe the dependencies in the system [10]–[12]. However, it is not yet well-investigated in a noisy neighbor situation.

Hence, the objective of this paper is to propose a solution in order to identify the noise source that disrupts and degrades the performance of virtual machines (VMs) or virtual network functions (VNFs, [2]) sharing the same virtualized infrastructure. The noise source can be either another VM or VNF. To achieve this purpose, we propose, in a first stage, an anomaly detection model that performs a clustering of normal behaviors of the machines. Any behaviour that does not belong to the normal space is considered as an abnormality in the system. In order to choose the suitable number of clusters, we introduce a parameter that allows selecting the best number of groups during clustering. In a second stage, RCA is proposed by creating a graph that describes the different dependencies in the system and the possible propagation paths of anomalies between the machines. Later, a score is defined to pinpoint

the machine that is responsible for the noise phenomenon. In order to evaluate our models, an experimentation study is held over a testbed that we created by using VMs under Openstack [13]. Real metrics from the different VMs are extracted using Prometheus [14].

This paper is organized as follows: Section II describes the related work. Section III presents the proposed noisy neighbor detection model as well as the RCA model. Section IV details the testbed and the real data experiments. Finally, Section V concludes the paper and it introduces some future works.

II. Related work

In literature, RCA has been studied for different types of anomalies but, up to our knowledge, it is not yet studied for a noisy neighbor situation. Several approaches use Bayesian Networks for the RCA such as in [9]. The model proposed in [11] studies anomaly alarms in general. It proposes an automatic RCA system that selects a shortlist of the most probable fault propagation sequences in component-based systems using a transaction call graph. The root cause is determined using frequent pattern mining which identifies the component that appears the most frequently in the graph and may be invoked in every alarm. Authors in [12] suggest a solution that detects more general network failures using machine learning and summarization techniques by means of an influence matrix that describes the causal relationship between different alarms. In their experiment, two use cases are studied, a cyberattack and a traffic migration. J. Lin et Al. in [10] present an automated anomaly detection method and RCA in virtualized cloud infrastructures, by investigating a DoS attack situation. Anomaly detection is performed by using an unsupervised learning using K-means algorithm. A component anomaly detection attached to each entity determines the clusters of normal behavior of the component and it pinpoints the locations of anomalies. Regarding the RCA, a set of anomaly propagation graphs is constructed. Every graph has a source component, and the rest of the entities are its nodes. The graph that has the best score in terms of reaching all of the components is the one that has the root cause. Their solution provides a well-defined architecture. However, the choice of the number of clusters as well as the determination of the anomalies after the clustering phase is not well-defined. Besides, in order to identify propagation paths, they empirically observe the usage of the different machines and they deduce the anomaly propagation paths with regard to a set of a predefined rules. This technique is time-consuming, complicated on a large scale and it needs human intervention.

III. Anomaly detection and root cause analysis for noisy neighbors

Given that several virtual machines are running over a same hypervisor, the objective of this section is to propose a solution for the detection of anomalous machines and for the pinpointing of the root cause of noisy neighbors. As described in Fig. 2, our solution is composed of four components (A-B-C-D). The first component (A) aims to describe the machines' behavior by collecting data using a node exporter agent in every machine. The collected counters are later sent to the second component (B) which is a monitoring system. This latter extracts metrics that describe the machine and it stores its historical usage in a dataset. Since in a noisy neighbor phenomenon, we may perceive a performance degradation of several machines, an anomaly detection module (C) is needed to run in every machine. Finally, we pinpoint the source of the noise by performing a RCA (D). This last component uses a classification in order to create a propagation graph that allows to localize the source of the noise.

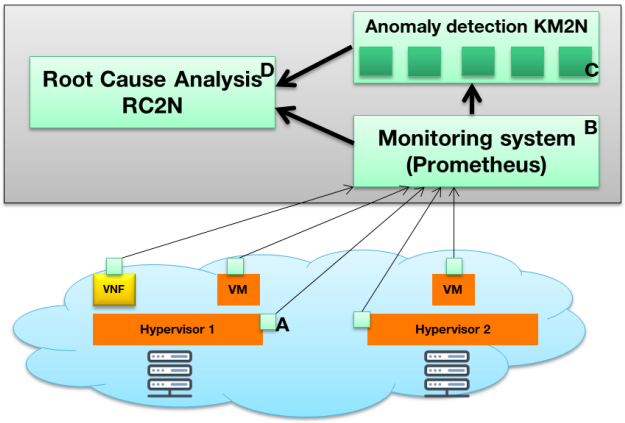

Fig. 2. Architecture of the proposed solution

A. K-Means for Noisy Neighbor: KM2N

Let x be the dataset collected from one machine and composed of N instances x_i that are collected periodically and $F = 3$ features belonging to $\{CPU\_usage, Network\_in, Network\_out\}$ so that $x_i = \{x_{i1}, .., x_{ij}\}$ where $1 \leq i \leq N$ and $1 \leq j \leq F$. CPU_usage is a percentage of the time during it the CPU was busy. Net_in is the amount of data arriving to the machine but originating elsewhere and Net_out is the amount of data originating at the machine to arrive elsewhere. Although memory usage could also be used, in our case, the memory is not shared so the machines do not compete over it. The collected data correspond to the normal usage and they are unlabeled. Thus, we use an unsupervised learning to explore the different patterns in the data. K-means algorithm [8] is a popular unsupervised technique that aims to partition the observations into K clusters. Although the algorithm proved its efficiency in different real applications, its drawback is the need to know a priori the number of clusters K. A big number of clusters may cause an overfitting and a high time execution and a small number may also mislead the training since the patterns risk to be combined. Hence, in order to select the model with the most appropriate number of clusters, we introduce a score $S(K)$ that allows making a trade-off between training performance, time execution and overfitting.
Given that ξ_k is the position of the centroid of the cluster

k and $x_i^{(k)}$ is the position of a data point i assigned to that cluster k, $S(K) = \sum_{k=1}^{K} \sum_{x_i^{(k)} \in cluster_k} ||\xi_k - x_i^{(k)}||^2$.

A new cluster can be added only if it achieves a significant gain g in terms of minimizing $S(K)$ i.e. the number of clusters increases from K to $K+1$ only if $S(K+1) \leq S(K) - g\% * S(K)$. The training phase of our model is summarized in Algorithm 1 where K^{max} is a fixed maximum number of clusters.

Result: K-means clustering with the best value of K
Initialize:
$g \leftarrow 30\%, diff \leftarrow 0, old \leftarrow 10^{10}, gain \leftarrow 0, K \leftarrow 1$;
while $K < K^{max}$ *or* $diff > gain$ **do**
  $K \leftarrow K + 1$;
  $KM2N = Kmeans(K)$;
  $diff \leftarrow old - S(K)$;
  $old \leftarrow S(K)$;
  $gain \leftarrow old * g$;
  if $diff > gain$ **then**
    $best \leftarrow K$;
  end
end
$KM2N = Kmeans(best)$;

Algorithm 1: Training phase

After the training phase, the different patterns in normal usage data are detected. A traditional K-means algorithm is unable to determine if a new data is anomalous since it aims to assign it to their nearest cluster. Therefore, in order to detect anomalies, we propose a threshold-based technique as detailed in Algorithm 2. A new data represents either a normal behavior, an anomaly, or a serious anomaly depending on its distance to the centroid. To do so, we first compute in every cluster k, the distance of every point $x_i^{(k)}$ belonging to k to its centroid ξ_k. Second, we normalize these distances using standardization so that: $z_i^{(k)} = \frac{||x_i^{(k)} - \xi_k|| - \mu_k}{\sigma_k}$, where μ_k and σ_k are respectively the mean and standard deviation of the distances of cluster k elements to its centroid ξ_k. The cluster distances will be rescaled so that they will have the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 1$. Third, for each new data x_{test}, a normalized distance $I(x_{test})$ from x_{test} to its nearest cluster $k_{nearest}$ is computed as follows: $I(x_{test}) = \frac{||x_{test} - \xi_{k_{nearest}}|| - \mu_{k_{nearest}}}{\sigma_{k_{nearest}}}$. Fourth, this distance is compared to a threshold θ as follows: if $I(x_{test}) \geq \theta$, then we have an anomaly, otherwise we have a normal behavior. θ has to be tuned during the test phase. A high value of θ means a high tolerance and allowing distant observations to be part of the normal behavior, while a low value increases the number of anomalies. Finally, the maximum distance I_{max} between the centroid and its training data allows to determine if the anomaly is serious or not.

Since some points in the data may represent outliers, we propose to use a smoothing technique over a sliding window of size sw that allows to remove outliers from the dataset and to make the patterns more visible.

Result: Decide whether x_{test} is an anomaly
Input: θ, x_{test};
$k_{nearest} \leftarrow argmin_k ||\xi_k - x_{test}||$;
$I(x_{test}) \leftarrow \frac{||x_{test} - \xi_{k_{nearest}}|| - \mu_{k_{nearest}}}{\sigma_{k_{nearest}}}$;
if $||x_{test} - \xi_{k_{nearest}}|| > \max_i ||x_i^{(k)} - \xi_{k_{nearest}}||$ **then**
  return(serious anomaly);
else
  if $I(x_{test}) > \theta$ **then**
    return(anomaly);
  else
    return(normal behavior);
  end
end

Algorithm 2: KM2N test phase

B. Root Cause analysis of Noisy Neighbors: RC2N

Once a noisy neighbor is detected, the root cause of the problem should be investigated. Hence, we propose a propagation path-based solution that observes the correlation between events and seeks the most probable anomaly propagation sequence. Propagation may occur following three scenarios: VM to PM, PM to VM and PM to PM where VM is a virtual machine and PM is a physical one. VM to VM propagations are avoided as we believe propagation cannot occur unless it passes through PMs. Besides, on a large scale, it is very complicated to monitor every $(VM - VM)$ couple in the infrastructure.

Let x^{RC} be the dataset composed of N^{RC} instances and of $F^{RC} = 6$ features so that $x_i^{RC} = \{x_{i1}^{RC}, ..., x_{ij}^{RC}\}$ where $1 \leq i \leq N^{RC}$ and $1 \leq j \leq F^{RC}$. The features are the CPU usage, the network inbound and the network outbound of two machines $M1$ and $M2$. Each instance is labeled according to the propagation path between these two machines that belongs to $\{PMtoVM, VMtoPM, PMtoPM\}$. In order to learn these propagation paths, we could use a multi-class classification with the desired three labels. However, a case where there is no propagation between the machines will not be predicted as the model will always try to find the closest class to the new data among the three labels. Besides, adding a new label "No propagation" to recognize the cases where the machines do not experience any propagation implies possessing data for all the couples that do not propagate anomaly to each other. This is unfeasible because of the variety of scenarios. Thus, we propose to use SVM one-class classifier [15] in order to train the propagation paths between each couple of machines. SVM one-class classifies new data as similar or different to the training set by identifying the smallest hypersphere that contains the majority of the training data [16]. Different models per propagation path will decide the type of the propagation for every new data as illustrated in Fig. 3.

Once propagation paths are trained, the next step consists of creating a propagation graph that represents the correlation between abnormal system observations. Let $G = (V, E)$ be a

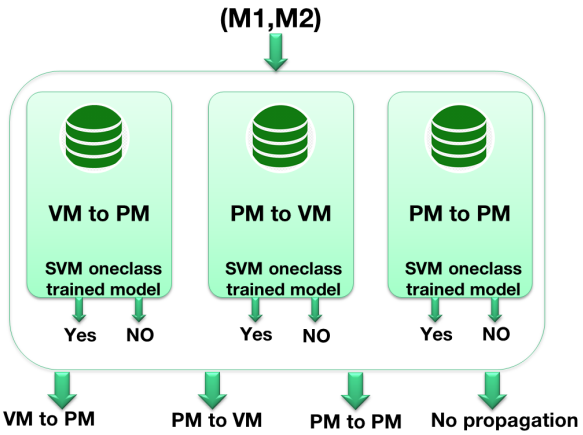

Fig. 3. RC2N_classifier for propagation path determination

direct graph, where V is the set of misbehaving components and E represents the propagation paths. In order to select the root cause, we assign a RCA score to each element of the graph. RCA score represents the ability of the component to propagate its anomaly to the rest of the affected entities. It is the minimum distance that the entity needs in order to reach every component n in the system. The score of a component c in a graph G is the following: $Score^{RCA}(c)_G = \sum_{n \in G} d_{c,n}$, where $d_{c,n}$ represents the shortest path distance from the component c to the component n using *Dijkstra* algorithm [17]. If n is not accessible by c then $d_{c,n} = \infty$ as detailed by Algorithm 3.

Result: Determine the root cause of noisy neighbor
Input: V, $Couples$;
Initialize: $E \leftarrow [\,]$, $G \leftarrow G(V, E)$;
for $couple \in Couples$ **do**
  if $RC2N\_classifier(couple) \in [VMtoPM, PMtoVM, PMtoPM]$ **then**
    $E \leftarrow E \cup \{couple\}$;
  end
end
for $c \in V$ **do**
  $Score^{RCA}(c)_G = \sum_{n \in V} d_{c,n}$
end
return$(argmin_c\{Score^{RCA}(c)\})$

Algorithm 3: RC2N algorithm where V is the set of machines, E is the set of propagation paths, $Couples$ is the set of couples of machines $(M1, M2)$ belonging to V.

IV. Experimental Study

Our testbed is set up in an *Openstack* environment as illustrated in Fig. 4. *Noise* is a server containing the victim machine suffering from noise and the noisy neighbor itself. *Asterisk* is an open source VoIP application and it represents our VNF deployed on a VM in order to receive calls. It is our victim. *Stress* is the noisy neighbor VM that performs intense calculations and consumes a lot of CPU. *P-tour04* is another server on which we deploy a traffic generator *SipP*. It is a free

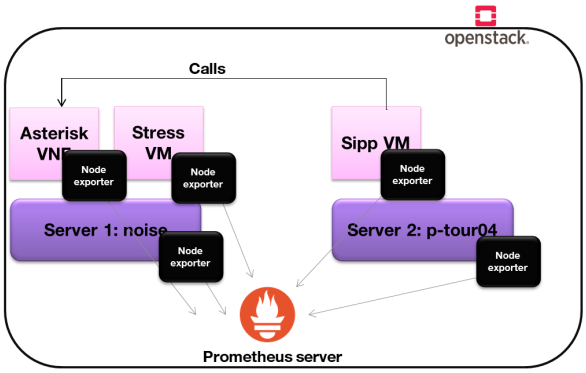

Fig. 4. Testbed of the solution

Open Source test tool for the SIP protocol. The role of this machine in the infrastructure is to generate calls to *Asterisk*. In the testbed, we use *Node Exporter* as an agent in every VM and PM. It allows collecting information from the machine in the form of counters that are sent to *Prometheus Server*. This server creates metrics that describe the utilization of the component in order to create a dataset.

A. KM2N training and test phases

The training data are created following different scenarios where components operate in different charges. Data are collected into *Prometheus* every 5 seconds, for about 8 days, for a total of 136920 instances per machine. In order to evaluate our model, several performance metrics [18] are used: (1) Accuracy is the ratio of correctly predicted data over total data; (2) Recall is the fraction of the elements of a class that are successfully predicted; (3) Precision is the fraction of correct predictions among all the predictions of a class and (4) F-score is a harmonic mean of precision and recall.

First, we need to identify the number of clusters K by computing the score $S(K)$. In order to evaluate the impact of the parameter g on the performance of anomaly detection, we determine the evolution of the number of clusters K with respect to g in *Asterisk* machine, and its effect on the performance of the model.

TABLE I
Performance metrics of Asterisk with respect to g for $sw = 50$ and $\theta = 10^{-3}$

$gain: g(\%)$	5	10	20	30	40	50
Nb of clusters K	11	9	6	6	6	2
Accuracy (%)	73	70	91.7	91.7	91.7	52.4
Positive recall (%)	61.1	55.5	96.5	96.5	96.5	89.3
Positive precision (%)	85.3	84.2	89.2	89.2	89.2	53.8
Positive Fscore (%)	71.2	67	92.7	92.7	92.7	67.2
Negative recall (%)	87.3	87.6	86	86	86	7.9
Negative precision (%)	65.2	62.1	95.3	95.3	95.3	38.4
Negative Fscore (%)	74.6	72.7	90.4	90.4	90.4	13.17

As illustrated in Table I, a very high gain (50%) is hardly reached so the retained number of clusters is 2. As for $g = 5\%$, we have 11 clusters because a significant gain is easily

achieved. In both cases, performance metrics are not the best. As shown in Fig. 5, for $g = 30$, $K = 6$ is retained since the score converges.

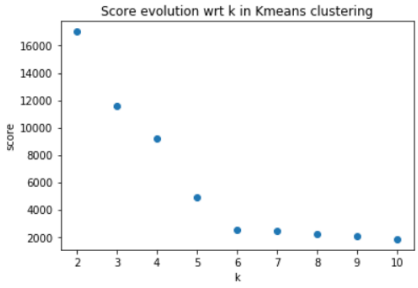

Fig. 5. Evolution of the score $S(K)$ with respect to K for K-means clustering

The trained clusters represent different patterns of normal usage of the machines.

The test phase aims to detect anomalies by identifying the closest cluster to every new data and by examining its distance to the centroid. An anomaly is decided by comparison with the threshold θ. A test scenario is created with unseen data in which normal scenarios are simulated where *Asterisk* machine receives normal calls from *sipP*. Noise scenarios are also generated where *Stress* machine starts to consume 100% of its allocated CPU in the *Noise* server and it degrades the performance of *Asterisk*.

Given that anomalies are positive results while normal data are negative results, Table II presents the results of the noisy neighbor detection with respect to the parameter θ that describes the strictness of the model towards anomalies. Several sliding windows have been testedμ. The window that gives the best results is of size $sw = 50$.

TABLE II
PERFORMANCE METRICS OF ASTERISK WITH RESPECT TO θ WITH $sw = 50$

θ	4	1	0.1	10^{-2}	10^{-3}	10^{-4}
Accuracy (%)	44.1	69.4	88.6	91.6	91.7	91.7
Positive recall (%)	0.1	52.7	90.4	96.2	96.5	96.5
Positive precision (%)	4.5	85.7	88.9	89.2	89.2	89.2
Positive Fscore (%)	0.2	65.2	89.6	92.6	92.7	92.7
Negative recall (%)	96.8	89.4	86.4	86	86	86
Negative precision (%)	44.7	61.2	88.2	95	95.3	95.4
Negative Fscore (%)	61.1	72.7	87.3	90.3	90.4	90.4

As shown in Table II, increasing θ results in a greater tolerance. For $\theta = 4$, positive F-score is very low as every new instance is considered as normal behavior. From $\theta = 10^{-3}$, the model gives more stable results, with an accuracy of 91,7%. Most of the noisy neighbor situations are detected with a positive recall=96.5%. If an alert is generated, the model is trustworthy with a precision up to 89.2% which means that the model does not generate a lot of false alarms. The normal usage of the machines is also detected with promising results which proves the efficiency of the model.

In Table III, we compare our unsupervised model KM2N to the results presented in [3] in which a supervised learning is applied with SVM and random forests (RF). In [3], similar features are used and also a similar testbed, except that Fibonacci is the machine that simulates the noise while it is Linux stress in our case. We notice that although KM2N

TABLE III
PERFORMANCE METRICS COMPARISON BETWEEN KM2N, SVM AND RANDOM FOREST (RF) [3]

Model	SVM [3]	RF [3]	KM2N
Precision (%)	86.16	92.32	96.5
Recall (%)	88.56	90	89.2
F1-score (%)	87.34	91.44	92.7

detects a little bit less anomalies than RF since it has a lower recall, it is still more precise and trustworthy since it generates fewer false alarms as it has a higher precision and a better F1-score metric.

A second comparison is applied by running several supervised models over the same data sets used by KM2N, extracted from our testbed. Three popular supervised techniques are used: SVM, RF and Multi-Layer Perceptron (MLP). The comparison is presented in Table IV.

TABLE IV
RESULTS COMPARISON BETWEEN KM2N, SVM, RF AND MLP

Model	SVM	RF	MLP	KM2N
Precision (%)	80	100	81	96.5
Recall (%)	88	99	100	89.2
F1-score (%)	84	99	89	92.7
Execution time	$>$ 1 hour	6.5 seconds	7.5 seconds	3.6 seconds

We notice that SVM model suffers from a very long execution time even with a linear kernel. Regarding random forests, given that the size of data is not immense and that we got a precision of 100%, we suspected that it is suffering from overfitting. This can be checked by applying on a larger volume of data. Multi-layer perceptron with 3 layers and 13 neurons in each layer is less precise than KM2N i.e it generates more false alarms.

Once a noisy neighbor situation is detected, the next step consists of launching the investigation of the whole infrastructure in order to identify all the involved machines.

B. RC2N training and test phases

The training data for RCA is collected from the couples of machines (*Stress*, *Noise*) labeled as *VMtoPM* and from the couple (*Asterisk*, *Noise*) labeled as *PMtoVM*. Once SVM one-class model is trained, the propagation paths between the machines are determined. The results are presented in Table V. It shows that the model succeeds to determine the propagation paths with an accuracy that is equal to 86%. Among all PM to VM propagations, 80% are detected. Besides, 78% of all VM to PM propagations are discovered by the model as well.

TABLE V
PERFORMANCE METRICS FOR PROPAGATION PATH DETERMINATION

Accuracy (%)	86	
Propagation	PMtoVM	VMtoPM
Recall (%)	80	78
Precision (%)	100	100
F1-score (%)	89	88

In Table VI, we compare one-class SVM to a similar one-class classifier called isolation forests [19].

TABLE VI
RESULTS COMPARISON BETWEEN ONE-CLASS SVM AND ISOLATION FOREST IN RC2N-CLASSIFIER

Model (%)	one-class SVM		Isolation forest	
Accuracy (%)	86		81	
Propagation type	PMtoVM	VMtoPM	PMtoVM	VMtoPM
Precision (%)	100	100	100	100
Recall (%)	80	78	64	80
F1-score (%)	89	88	78	89

The comparison shows that although one-class SVM gives similar performances as isolation forests regarding VMtoPM isolation, it still gives better results regarding PMtoVM propagation.

After determining the propagation paths, we create a direct graph that represents reachable components through propagation paths, using Algorithm 3. Later, a RCA score is calculated in every node to describe the ability of that node to propagate its anomaly to the rest of the machines: 3 for *Stress*, ∞ for *Asterisk*, and ∞ for *Noise*. Both *Asterisk* machine and *Noise* server have infinite scores so they are unable to propagate their anomaly to all the machines. However, *Stress* machine gives a finite score so it represents the root cause of our problem, it is then the noisy neighbor. ¨

This experimental study proves that our model is capable of detecting anomalies in the VNF by comparing its usage to historical data. Morever, the problem can be solved not by migrating the VNF to another server, as this is not always reliable, but by pinpointing the root cause of the problem and by asking the cloud manager to solve the issue from its source.

V. CONCLUSION AND FUTURE WORKS

Thanks to 5G technology, network functions will no longer be tightly connected to the hardware on which they operate. Therefore, network becomes more agile. Nevertheless, cloud computing systems continue to provide poor isolation, so noisy neighbor phenomenon remains a major problem. Not detecting and correcting this anomaly in real time may result in some economic losses at the cloud provider level. This work highlights the noisy neighbor issue and it proposes a complete solution that allows performing a RCA in order to determine the source of the noise. To do so, a testbed in *Openstack* environment is created. An anomaly detection model KM2N is proposed to catch anomalies in the infrastructure. A RCA is later performed by creating RC2N model that operates using a propagation graph. Experimentation was held on real data and the results are promising which prove the efficiency of the proposed approach.

Further work should include testing other types of anomaly such as overload and in more complex infrastructure. It would be also interesting to test the model on other VNFs (different from *Asterisk*) because only system metrics are used in our model so it is not specific to one application. Finally, another future direction is to propose an automated solution for tuning the model's parameters.

REFERENCES

[1] "Ericsson Mobility Report," Ericsson, Tech. Rep., june 2019. [Online]. Available: https://www.ericsson.com/49d1d9/assets/local/mobility-report/documents/2019/ericsson-mobility-report-june-2019.pdf

[2] NFV White Paper, "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. Issue 1," Oct. 2012.

[3] U. Margolin, A. Mozo, B. O. Rubio, D. Raz, E. J. Rosensweig, and I. Segall, "Using machine learning to detect noisy neighbors in 5g networks," *CoRR*, vol. abs/1610.07419, 2016.

[4] B. Ordozgoiti, A. Mozo, S. G. Canaval, U. Margolin, E. J. Rosensweig, and I. Segall, "Deep convolutional neural networks for detecting noisy neighbours in cloud infrastructure," in *ESANN*, 2017.

[5] P. Veitch, E. Curley, and T. Kantecki, "Performance evaluation of cache allocation technology for nfv noisy neighbor mitigation," *2017 IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–5, 2017.

[6] S. H. Nikounia and S. Mohammadi, "Hypervisor and neighbors' noise: Performance degradation in virtualized environments," *IEEE Transactions on Services Computing*, vol. 11, no. 5, pp. 757–767, Sep. 2018.

[7] C. Wang, V. Talwar, K. Schwan, and P. Ranganathan, "Online detection of utility cloud anomalies using metric distributions," *2010 IEEE Network Operations and Management Symposium - NOMS 2010*, pp. 96–103, 2010.

[8] E. Alpaydin, *Introduction to Machine Learning*. The MIT Press, 2014.

[9] M. Ruiz, F. Fresi, A. P. Vela, G. Meloni, N. Sambo, F. Cugini, L. Poti, L. Velasco, and P. Castoldi, "Service-triggered failure identification/localization through monitoring of multiple parameters," in *ECOC 2016; 42nd European Conference on Optical Communication*, Sep. 2016, pp. 1–3.

[10] J. Lin, Q. Zhang, H. Bannazadeh, and A. Leon-Garcia, "Automated anomaly detection and root cause analysis in virtualized cloud infrastructures," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, April 2016, pp. 550–556.

[11] K. Wang, C. Fung, C. Ding, P. Pei, S. Huang, Z. Luan, and D. Qian, "A methodology for root-cause analysis in component based systems," in *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, June 2015, pp. 243–248.

[12] J. M. N. Gonzalez, J. A. Jimenez, J. C. D. Lopez, and H. A. P. G., "Root cause analysis of network failures using machine learning and summarization techniques," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 126–131, Sep. 2017.

[13] "Openstack open source cloud computing software," https://www.openstack.org/, accessed: 2019-10-02.

[14] "Prometheus overview," https://prometheus.io/docs/introduction/overview/, accessed: 2019-10-02.

[15] P. Oliveri, "Class-modelling in food analytical chemistry: Development, sampling, optimisation and validation issues – a tutorial," *Analytica Chimica Acta*, vol. 982, 05 2017.

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

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[18] N. Japkowicz and M. Shah, *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, 2011.

[19] F. T. Liu, K. M. Ting, and Z. hua Zhou, "Isolation forest," in *In ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining. IEEE Computer Society*, pp. 413–422.