Machine Learning based SLA-Aware VNF Anomaly Detection for Virtual Network Management

Jibum Hong, Suhyun Park, Jae-Hyoung Yoo, and James Won-Ki Hong
Department of Computer Science and Engineering, POSTECH, Pohang, Korea
{hosewq, sh.park11, jhyoo78, jwkhong}@postech.ac.kr

Abstract—**Since the concept of Software-Defined Networking (SDN) and Network Function Virtualization (NFV) has been proposed, telcos and service providers have leveraged these concepts to provide their services more efficiently. However, as the virtual network in the data centers becomes more complex, a variety of new network management problems arise. To deal with these management problems, it is necessary to monitor and analyze resource usage and traffic load of Virtual Network Functions (VNFs) operating on the virtual network. Recently, there have been many attempts to develop technologies that enable network management without human intervention. In this paper, we specify our anomaly detection problem with scenarios involving SLA violations to satisfy the practical needs of network management. Also, we set the real-world NFV environment to generate anomalous data corresponding to each scenario and extend our approach to implementing the system for root-cause localization which identifies the exact VNF instance causing the SLA-related anomalies. We use the datasets collected from the VNFs' service function chain scenarios implemented on OpenStack environment, and compare the accuracy of the anomaly detection models generated by various machine learning algorithms. Our experimental results show the best model has F1-measure over 95% for anomaly detection and 93% for root-cause localization.**

Index Terms—**anomaly detection, root-cause localization, network monitoring, machine learning, NFV management**

I. Introduction

Software-Defined Networking (SDN), Network Function Virtualization (NFV), and Network Virtualization (NV) are giving us new ways to design, build, and operate networks. With these technologies, telcos and service providers can reduce CAPEX and OPEX by replacing the closed network functions to softwarized Virtual Network Functions (VNFs). In addition, cloud computing combines these technologies to use computing resources more efficiently in data centers, and provides flexibility and agility for application service deployment and management [1].

With the wide-spread use of these technologies, the number of VNFs and services operating on virtual networks is increasing significantly. In contrast, these increases can complicate virtual network operations, and may cause performance degradation, service failures, or system overload problems. For this reason, the importance of virtual network management in NFV environment is emphasized.

To solve the management problem in a virtual environment, several requirements [2] are defined regarding orchestration, performance, security, etc. In particular, resource management and fault management techniques have been developed. Resource management is to provide optimized computing and network resource for VNFs such as the allocation or scaling of virtual resources. Fault management is to recover the system or network from failures, or to detect abnormal behaviors.

Network administrators can handle these tasks well in small networks, but in the case of large-size networks and the networks that have complex dependencies, it requires different approaches. Therefore, recent research on virtual network management using machine learning and deep learning techniques has attracted much attention to solving the above problems. Several attempts have tried to develop technologies that enable the network to understand its status and optimally manage the network without human intervention [3].

We focus on the anomaly detection technique for virtual networks in NFV environment as a fault management measure. However, the existing studies are mostly based on binary classification for detecting VNFs' anomalies related to performance bottlenecks such as CPU/memory usage, and throughput. In this paper, we propose a machine learning-based VNF anomaly detection and root-cause localization system, considering the system resource usage and service level agreement (SLA) violation status both. We use OpenStack testbed to operate various VNFs in real-world topology. And we derive and deploy the optimal machine learning model by testing and analyzing the data collected from the testbed with various machine learning models and comparing their accuracy.

II. Background and Related Work

A. Background

Traditionally, network management is performed manually by human administrators. However, as network and service requirements become more diverse and complex, deploying or upgrading the services takes a much longer time, and the demands for human experts for detailed network configuration increase network operation costs. To solve these problems, the development of network management technology using machine learning is attracting great attention [4].

In this trend, automating the virtual network management in NFV environment using machine learning generally consists of the processes as shown in Fig. 1. First, VNFs operate on the NFV Infrastructure (NFVI) by using its virtual resources, and Analytics monitors VNF resource usage. ETSI NFV Working Group presents representative monitoring data that can be collected and used in an NFV environment (e.g. CPU utilization,

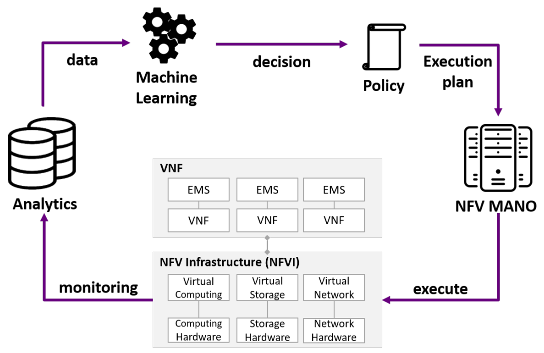


Fig. 1: Overall process of ML-based NFV management

memory usage, traffic load, etc.) [5]. Then, the monitoring data are stored in the database. Next, the data are converted into datasets to train the machine learning algorithm, and the algorithm generates a model which has a specific purpose. After that, the model creates policies on how to operate the virtual network based on the current states. Finally, NFV Management and Orchestrator (MANO) operates and manages the entire NFV environment according to the policies.

Among the requirements for network management automation, our work focuses on VNF anomaly detection, which is one of the processes for virtual network management automation in NFV environment. We detect abnormal states of VNFs and the location of abnormal VNFs based on system and network resource usage by training monitoring data through machine learning algorithms when performance bottlenecks and SLA violations occur.

B. *Related Work*

There exist many anomaly detection studies in various fields of network management, but most of the studies are different from each other in their settings of network environments and the definition of anomalies. E. Chuah et al. [6] investigates the effect of high resource usage on system failures using resource usage monitor and log analysis. In addition, it is very difficult to get datasets related to abnormal situations because they happen rarely and unexpectedly as the term ”anomaly” itself implies. So many studies use fault injection techniques to generate the software or hardware faults [7], [10], [12], [14].

To automatically detect anomalies, statistical solutions have been developed. J. Hochenbaum et al. [7] proposes statistical measures based on the three-sigma rule, moving averages, and the Seasonal Trend decomposition using Loess (STL) algorithm and compares their efficiencies of detecting anomalies in cloud infrastructure data. C. Wang et al. [8] proposes a method called EbAT which detects anomalies by analyzing the distribution of arbitrary metrics instead of the thresholds of individual metrics, and compares its performance with that of the threshold-based method. J. Chen et al. [9] proposes a MADEL algorithm, which uses a matrix differential decomposition (MDD) based anomaly detection and localization in NFV networks through round-trip time (RTT) and packet loss rate. These statistical approaches might be efficient for automatic anomaly detection when the anomaly could be defined by a single threshold value and the threshold could be set clearly. However, statistical technologies can not classify those anomalies which are caused by more complicated conditions.

As more and more attempts to apply artificial intelligence to network management have increased, many studies use machine learning techniques to detect anomalies, especially those related to the performance of VNFs in NFV environment. C. Sauvanaud et al. [10] proposes the anomaly detection and root-cause localization model which classifies the states of VNFs into normal and abnormal, with Random Forest (RF) algorithm in vIMS environment. PREPARE system [11] provides automatic performance anomaly prevention for virtualized cloud computing infrastructures by integrating the Markov chain model with the Tree Augmented Naive Bayes (TAN) algorithm. Instead of providing a specific machine learning model, J. Qiu et al. [12] applies Support Vector Machine (SVM), Decision Tree (DT), RF, and Neural Network (NN) to detect the performance anomalies, and compares the performance of each model. H. Bouattour et al. [13] proposes the anomaly detection and root-cause analysis method based on k-means clustering and a one-class SVM classifier by evaluating the proposed model with VoIP application. And BARCA [14] framework detects the distributed system’s abnormal behaviors such as deadlock and memory leak through online SVM classifiers, and compares the performance of various feature extraction methods.

Major differences compared to related work are as follows. Firstly, most of the existing studies detect the abnormal states with binary classification related to typical resource overloads such as CPU and memory. In contrast, our work detects SLA-related abnormal states with higher accuracy than existing studies. Secondly, this work provides the root-cause location of VNF which causes the SLA violations. Thirdly, our work collects large data for service function chaining scenarios, while some existing studies use a relatively small number of data for training and validation.

III. Methodology and Implementation

A. *Methodology*

Fig. 2 illustrates an overview of the proposed anomaly detection method. The proposed method consists of 3 main processes: virtual network monitoring, preprocessing, and training models. We use supervised learning algorithms to learn the relationship between feature data and labeled data. Through the main processes, we choose the best model by comparing the performance among the models.

1) Virtual Network Monitoring: To train the anomaly detection model, we have to monitor the virtual network operating on NFV Infrastructure. Monitoring functions for the virtual network generally consist of monitoring agents, a monitoring service, and a dashboard.

Monitoring agents collect the status data of each VNF in the virtual network. Monitoring metrics collected by the agents are

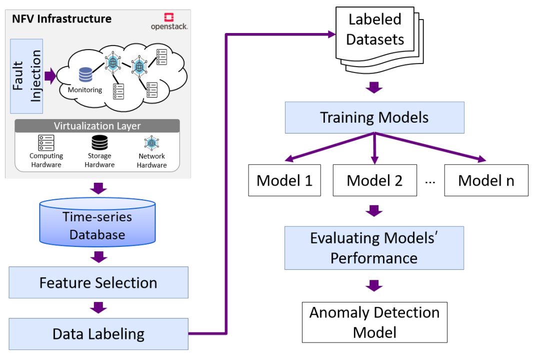


Fig. 2: Proposed ML-based anomaly detection method

the subsets of representative metrics [6] such as CPU utilization, memory usage, and traffic load, etc. Monitoring agents then transfer the data to the monitoring service, which stores the collected data into the time-series database. To provide visibility, a simple dashboard is implemented. The stored data are transformed into the datasets for model training.

When collecting the data from the virtual network, because anomalies do not occur frequently on the network, we use fault injection techniques to generate abnormal states with precise controllability. In particular, we emulate the various software or hardware faults which exist in the system: 1) generating the abnormal states to VMs where VNFs operate, and 2) generating heavy workload which does not guarantee the correct service such as sending tremendous network traffic.

The first method causes faults directly into the VM where the VNF operates. The fault situations used in this method are considered in terms of CPU utilization, memory usage, disk I/O access, network latency, and network packet loss. The second method causes heavy overload to the network through tremendous traffic. This may cause packet processing delay and packet drop by the kernel. We emulate a large amount of traffic to VNFs and a large number of accesses to web servers which pass through the VNFs.

2) Preprocessing: Preprocessing converts the monitoring data collected through the previous processes into a suitable form to train models. This process consists of feature selection and data labeling (Fig. 2). First, feature selection discriminates the metrics which are most relevant to the criteria for distinguishing abnormal states with the metrics collected through monitoring. Then we remove redundant metrics that are intrinsically correlated to each other. In this process, we extracted the 25 features (described in Table I) for instance information and resource usage of VNFs through feature selection and used the features to train models.

Data labeling is a process to distinguish the extracted feature data into normal and abnormal to train the models for supervised learning-based machine learning algorithms. In the case of root-cause localization function, the abnormal states are labeled more specifically with the location of VNF which shows abnormal behaviors. However, simply defining abnormal states as cases in which the metrics such as CPU utilization are temporarily increased for a very short time causes many false alarms. So we define abnormal states in 2 cases: 1) VNF performance bottleneck and 2) SLA violations.

TABLE I. Selected features for anomaly detection

Features	Description	Features	Description
time	Measurement time	mem_used	Memory - used space
instance	VNF instance name	disk_free	Disk - free space
cpu_idle	CPU - idle time	disk_reserved	Disk - reserved space
cpu_interrupt	CPU - interrupt time	disk_used	Disk - used space
cpu_nice	CPU - nice status time	io_read	I/O - read bytes
cpu_softirq	CPU - softirq time	io_write	I/O - write bytes
cpu_steal	CPU - stolen time	io_time	I/O - spent time
cpu_system	CPU - used by kernel mode	network_rx_bytes	Received traffic bandwidth
cpu_user	CPU - used by user mode	network_tx_bytes	Transmitted traffic bandwidth
cpu_wait	CPU - I/O wait time	network_rx_packets	Received traffic bandwidth
mem_free	Memory - free space	network_tx_packets	Transmitted traffic bandwidth
mem_buffered	Memory - buffered space	network_latency	Hop latency between VNFs
mem_cached	Memory - cached space		

VNF performance bottleneck is represented as packet drops occurring inside the VNFs due to the lack of available system resources caused by the fault injection techniques. The packet drop rate is calculated by comparing the number of incoming and outgoing packets processed by the VNFs. We labeled the data as abnormal states when VNFs' packet drops occurred over 0.1% by fault injection, and labeled the rest as normal.

SLA violations are different for each service, but they generally include average service time (response time) and service request failure rate for operating service. For example, web services require an average response time between 0.5 or 1 second, and availability between 99% or 99.9% [15].

3) Training Models: There are various kinds of supervised learning algorithms to solve classification problems. To choose the best anomaly detection algorithm, trained models need to be tested with the given dataset. Among tens of models showing the highest performance in a training process, in most of the cases, Distributed Random Forest (DRF) [16], Gradient Boosting Machine (GBM) [17], Extreme Gradient Boost (XGBoost) [18], Deep Learning (FNN) [19] based models are included. First, in the case of DRF model, we adjust hyperparameters such as the number of trees, depth of trees, and the number of leaves to optimize the model by minimizing log loss. And in the case of GBM and XGBoost models, the hyperparameters are similar to DRF except that we additionally adjust the row/column sample rate per tree which doesn't exist for DRF. Besides, in the case of XGBoost model, one-hot encoding method is used for categorical features. Lastly, in the case of the Deep Learning model, we adjust the number of nodes for the two hidden layers with ReLU (Rectified Linear Unit) activation function, and then we apply dropout to the neural network to regularize it.

B. Implementation

Fig. 3 illustrates the implemented architecture of the anomaly detection system's prototype based on the generated model through the proposed method. The virtual network is configured on the physical environment by using OpenStack environment, and operates various services on this NFV Infrastructure. First, VNFs and network status data are collected and transferred to the database and dashboard. Next, the anomaly detection function imports the real-time data from the database and preprocesses the data for the deployed anomaly detection module (based on Python) on the system. Then, the anomaly detection module detects the abnormal status by identifying the processed real-time data, and localizes the root-cause VNF which has the anomalies. Finally, the anomaly detection module notifies the alert to a network administrator or NFV orchestrator to operate and manage the virtual infrastructures well. To deploy the anomaly detection module, we trained the anomaly detection models. The training processes consist of data collection part and data analysis part.

1) Data Collection: Data collection is to collect the data from VNFs for anomaly detection model training. It is the implementation of virtual network monitoring and fault injection processes in our methodology. We first build VNFs in our OpenStack-based NFVI environment [20] to configure the VNFs' service scenarios. Each scenario emulates the abnormal state of the VNFs through fault injection techniques as well as the normal operation of the VNFs. In each fault injection technique, stress-ng [21] injects faults related to system resource usages used by the VMs, such as high CPU utilization, lack of memory, and disk accesses anomalies. Also, Linux tc [22] generates network latency (packet delay) and packet loss to perform fault injection related to network anomalies and SLA violations.

To monitor the VNFs' normal and abnormal states, we implemented the virtual network monitoring functions. Collectd [23] which is a monitoring daemon agent monitors the status of each VM which VNFs operate (e.g. resource usage, traffic load, etc.). Then, the monitoring data are stored in InfluxDB [24], a time-series database. Grafana [25] dashboard provides the data stored in the database in the form that users request by sending queries (InfluxQL) to InfluxDB.

2) Data Analysis: Data analysis includes preprocessing and model training steps. In this part, we train the anomaly detection models with the exported data. Among the metrics of VNFs' historical data which are exported from Grafana, we extract 25 features according to the feature selection process (Table I). Then, we label the extracted feature data as normal and abnormal states. Data labeling is performed based on the occurrence of SLA violations and each VNF's abnormal behaviors for root-cause localization.

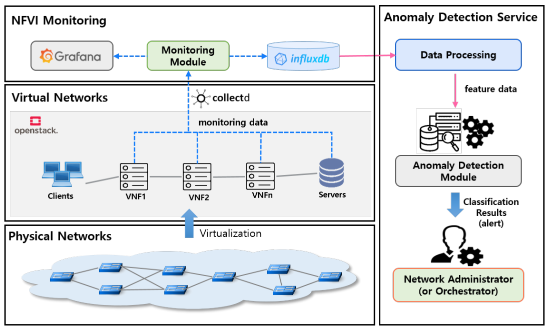


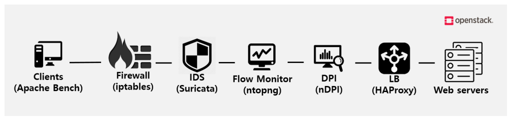
Fig. 3: Overall architecture of proposed system

The labeled dataset created by the preprocessing process guides the models to learn the relationships between feature data and labeling data through supervised learning-based machine learning algorithms. In this part, we use the R language and H_2O framework [26] to train anomaly detection models. Among the many algorithms, we implement 4 algorithms (XGBoost, GBM, DRF, and Deep Learning), which show the highest performance in our experiments, by controlling each model's hyper-parameters. After each model is generated, we compare the performance of each anomaly detection model and finally select the most suitable model.

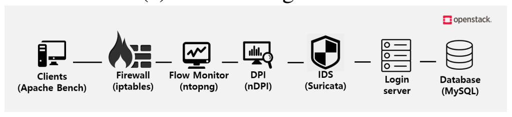
IV. Performance Evaluation

A. Experimental Setup

To evaluate the performance of the proposed anomaly detection method, we first set up the experimental testbed environment [20] with similar topology to that of a Multi-access Edge Computing (MEC) scenario. Fig. 4 illustrates our experimental testbed. In the testbed, we used the OpenStack (Rocky release) to construct a virtual network with a monitoring system, and VNFs' service function chainings (SFCs) in a web hosting service and login authentication scenarios. The SFCs consist of open-source VNFs: firewall (FW, iptables [27]), intrusion detection system (IDS, Suricata [28]), flow monitor (FM, ntopng [29]), deep packet inspection (DPI, nDPI [30]), and load balancer (LB, HAProxy [31]).



(a) web hosting service



(b) login authentication

Fig. 4: Experimental testbed setup based on OpenStack: web hosting service scenario (a), login authentication scenario (b)

In the web hosting service scenario (Fig. 4a), the client is to pass through the 5 types of VNFs to access the web servers, and in the login authentication scenario, the client sends login requests to the login server which are connected to remote database (Fig. 4b). We emulated HTTP requests through a web stress tool (Apache Bench) to generate the client-side HTTP traffic pattern for web servers.

B. Experiment 1: Web Hosting Service

At first, the clients send the HTTP requests to web servers in our testbed. We configured the number of connections as 10 to 500, and the number of HTTP requests as 1,000 to 50,000 for making traffic patterns. In this experiment, we collected about 90,000 data which consists of 60% normal data and 40% abnormal data by monitoring every second. In labeling

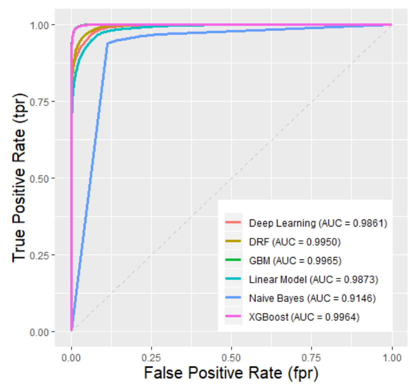


Fig. 5: ROC curves of experiment 1

SLA violation metrics, we labeled the abnormal data with the reduced average service time (to 250 ms) and availability (to 99.95%, represented as a service request failure rate over 0.05%). Because our testbed which resembles the MEC topology is relatively smaller than the real MEC environment, we adjusted the SLA violation standards [24] by applying more strict ones which are adequate for the testbed size. Finally, to evaluate the trained models' performance more precisely, we gave weight to abnormal data to balance the data classes, and compared each model's performance through 5-fold cross-validation.

We compare each model's ROC (Receiver Operating Characteristic) curves (Fig. 5). GBM and XGBoost model show the best performance of 0.996 AUC (Area Under the Curve) value. DRF model follows the GBM and XGBoost models by a narrow margin with the AUC value of 0.995. Also, we compare the mean values of precision, recall, F1-measure, and training time from the cross-validation process. For all models of selected top 4 algorithms, the labeled data are classified with an F1-measure of higher than 0.9. As shown in Table II-a, GBM and XGBoost show the best performance, followed by DRF and Deep Learning in order. In the case of XGBoost and GBM, the F1-measures are over 0.96, but GBM requires more training time.

TABLE II. Experimental results of web hosting service

(a) Prediction results of trained models

Algorithms	Precision	Recall	F1-Measure	Training Time
Deep Learning	0.9366	0.9265	0.9314	29.47s
DRF	0.9521	0.9537	0.9528	11.46s
GBM	**0.9592**	**0.9647**	**0.9619**	19.96s
XGBoost	0.9581	0.9626	0.9603	**5.03s**

(b) Prediction results of XGBoost with root-cause localization

Datasets	Measures	Caused in FW	Caused in IDS	Caused in FM	Caused in DPI	Caused in LB	Caused by traffic load
With hop-by-hop Latency	Precision	0.9541	0.9519	0.9711	0.9799	0.9637	0.9708
	Recall	0.9584	0.9375	0.9693	0.9882	0.9744	0.9479
	F1-Measure	**0.9562**	**0.9446**	**0.9702**	**0.9841**	**0.9690**	**0.9592**
Without hop-by-hop Latency	Precision	0.9008	0.9252	0.9102	0.9511	0.9611	0.9306
	Recall	0.9713	0.9487	0.9833	0.9928	0.9860	0.9626
	F1-Measure	**0.9347**	**0.9368**	**0.9454**	**0.9715**	**0.9734**	**0.9464**

Based on these results, we apply the XGBoost algorithm which is used for the best performing model considering training time and accuracy to root-cause localization prediction. The abnormal data are labeled into more specific VNFs and heavy traffic loads which cause the SLA violations. As shown in Table II-b, the 2 kinds of datasets are used for validation. For overall labels, The dataset which includes the hop-by-hop latency feature which can be collected by our testbed shows slightly better performances than the dataset without the hop-by-hop latency feature. Among the VNFs which consist of SFC in this experiment, prediction results for the IDS anomalies are the worst (0.94), while DPI shows the best prediction results (0.98).

C. *Experiment 2: Login Authentication*

In the login authentication scenario (Fig. 4b), the configurations for traffic patterns, monitoring methods, and the model training processes are the same as in the web hosting scenario. In this experiment, we collected about 120,000 data and labeled this dataset into two ways: 1) 250 ms of avg. service response time, 99.95% of availability, and 2) 200 ms of avg. service response time, 99.99% of availability. Unlike web hosting scenario which includes media data, the requests in this scenario require only login authentication. We could show the avg. service response time and request failure rate are less than the values in the previous scenario. So we labeled the abnormal data with more strict standards, and compared the results of each model and labeling way.

Like web hosting scenario, we first compare each model's performance through the ROC curve (Fig. 6). The results of 2nd labeling case show slightly lower performance than 1st labeling case. But in both labeling cases, GBM and XGBoost model show the best performance at about 0.99 AUC values. Also, for all models of selected top 4 algorithms, the labeled data are classified with an F1-measure of higher than 0.9 (90%). In both labeling cases, GBM and XGBoost show the best performance, followed by DRF and Deep Learning in order (Table III-a). XGBoost and GBM models show the F1-measures over 0.95 in 1st labeling case and 0.93 in 2nd labeling case.

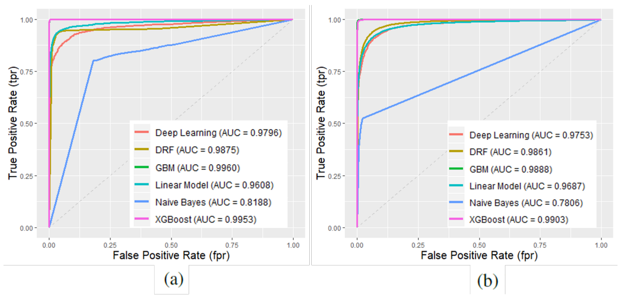


Fig. 6: ROC curves of experiment 2: (a) labeled with 250ms avg. response time and 99.95% availability, (b) labeled with 200ms avg. response time and 99.99% availability

TABLE III. Experimental results of login authentication

(a) Prediction results of trained models

Datasets	Algorithms	Precision	Recall	F-Measure	Training Time
250ms resp. time, 99.95% availability	Deep Learning	0.9178	0.8887	0.9027	40.50s
	DRF	0.9403	0.9326	0.9363	11.61s
	GBM	**0.9555**	0.9487	0.9520	19.45s
	XGBoost	0.9505	**0.9556**	**0.9530**	**9.45s**
200ms resp. time, 99.99% availability	Deep Learning	0.9143	0.9011	0.9077	34.84s
	DRF	0.9325	0.9195	0.9259	15.93s
	GBM	0.9345	0.9308	0.9327	23.88s
	XGBoost	**0.9390**	**0.9372**	**0.9381**	**12.52s**

(b) Prediction results of XGBoost with root-cause localization

Measures		Measures	Caused in FW	Caused in FM	Caused in DPI	Caused in IDS	Caused by traffic load
250ms resp. time, 99.95% availability	With hop-by-hop Latency	Precision	0.9590	0.9675	0.9677	0.9675	0.9916
		Recall	0.9250	0.9357	0.9434	0.9357	0.7600
		F1-Measure	**0.9417**	**0.9513**	**0.9554**	**0.9513**	**0.8605**
	Without hop-by-hop Latency	Precision	0.9614	0.9770	0.9620	0.9770	0.9966
		Recall	0.9297	0.9531	0.9497	0.9531	0.8915
		F1-Measure	**0.9453**	**0.9649**	**0.9558**	**0.9649**	**0.9411**
200ms resp. time, 99.99% availability	With hop-by-hop Latency	Precision	0.9519	0.9559	0.9648	0.9559	0.8989
		Recall	0.9482	0.9541	0.9575	0.9541	0.8973
		F1-Measure	**0.9500**	**0.9550**	**0.9612**	**0.9550**	**0.8981**
	Without hop-by-hop Latency	Precision	0.9404	0.9434	0.9404	0.9434	0.8989
		Recall	0.9542	0.9594	0.9542	0.9594	0.8937
		F1-Measure	**0.9472**	**0.9513**	**0.9472**	**0.9513**	**0.8963**

Also, based on these results, we apply the XGBoost algorithm which is used for the best performing model to root-cause localization prediction. As shown in Table III-b, we used the 2 kinds of datasets (with and without hop-by-hop latency feature) in each labeling case like the web hosting scenario. In the aspect of labeling cases, the overall results of the 1st labeling case (250 ms of avg. response time and 99.95% of availability) is slightly worse than the results of the 2nd labeling case (200 ms of avg. response time and 99.99% of availability). In the aspect of hop-by-hop latency feature, the datasets which include the hop-by-hop latency feature show better performances than the datasets without hop-by-hop latency feature. The cause of heavy traffic overload has the worst prediction results about 0.89 of F1-measure.

D. Discussion

While the target environments and the definition of anomalies are different, the attained values from our experiments are similar or relatively higher than any other results of related work. In all experiments, XGBoost models show the best classification accuracy with the shortest training time. GBM model also shows good performance, but it takes longer to train with a high risk of overfitting. In the aspect of feature importance, the hop-by-hop latency feature was the most important feature than other features, and the metrics related to CPU, network bandwidth, memory, and disk I/O follow.

Also, for the generated model's generalizability, we validated the generated anomaly detection model with the test datasets of different traffic patterns. The test datasets consist of about 20,000 normal data and 20,000 abnormal data for each scenario by random sampling. The results show about 91% and 90% of F1-measure for the two scenarios (Table IV-a). However, in root-cause localization results, IDS and LB show similar prediction results, but other VNFs (FW, FM, and DPI) and the anomalies caused by heavy traffic load cases show lower accuracy. To solve this problem, we need to improve the model's performance or consider individually training anomaly detection models for each VNF.

TABLE IV. Prediction results with the test datasets

(a) Prediction results of trained models

Service Scenario	Accuracy	Precision	Recall	F-Measure
Web Hosting	0.9370	0.9336	0.8994	0.9162
Login Authentication	0.8958	0.9452	0.8675	0.9047

(b) Detailed prediction results of web hosting scenario

Measures	Caused in FW	Caused in IDS	Caused in FM	Caused in DPI	Caused in LB	Caused by traffic load
Precision	0.9802	0.9488	0.9663	0.9919	0.9947	0.9288
Recall	0.8417	0.9119	0.8293	0.8735	0.9686	0.8489
F1-Measure	**0.9056**	**0.9300**	**0.8925**	**0.9289**	**0.9815**	**0.8870**

(c) Detailed prediction results of login authentication scenario (200ms response time, 99.99% availability case)

Measures	Caused in FW	Caused in FM	Caused in DPI	Caused in IDS	Caused by traffic load
Precision	0.8562	0.8883	0.9037	0.9300	0.8565
Recall	0.9275	0.9132	0.8890	0.9400	0.8592
F1-Measure	**0.8904**	**0.9006**	**0.8963**	**0.9350**	**0.8578**

To apply the anomaly detection model in more dynamic environments, we consider improving our model to deal with various service scenarios. And to include system or application-level failures, syslog data or error messages generated by applications could be used for labeling. By utilizing the proposed anomaly detection function, it could help VNF lifecycle management decision making such as VNF auto-scaling and migration functions for network administrators (or NFV orchestrators).

V. Conclusion

The current NFV environment faces various demands for efficient management and operation in a fast-changing network. In this paper, we propose a machine learning-based VNF anomaly detection with the root-cause localization system as a step to automate virtual network management in NFV environment. We configure the two SFC scenarios in an OpenStack-based testbed with real-world topology, and evaluate the anomaly detection model trained by the proposed method. The results of our experiments show that the F1-measure of the most suitable anomaly detection model is over at least 95%, and 93% of the root-cause localization accuracy except for the heavy traffic load case in login authentication.

For future work, we will extend to our anomaly detection model to apply more various use-cases such as vIMS environment or other application services. And we will extend our model to log analysis to more accurately define and detect anomalies.

ACKNOWLEDGMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2018-0-00749, Development of virtual network management technology based on artificial intelligence).

REFERENCES

[1] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Open-stack: toward an open-source solution for cloud computing," in International Journal of Computer Applications, vol. 55, no. 3, pp. 38-42, Oct. 2012.

[2] IRTF Network Function Virtualization Research Group, "Network Functions Virtualisation – Update White Paper," [Online]. Available at https://portal.etsi.org/NFV/NFV_White_Paper2.pdf.

[3] R. Boutaba et al., "A comprehensive survey on machine learning for networking: evolution applications and research opportunities", in Journal of Internet Services and Applications, vol. 9, issue 1, 2018.

[4] D. Rafique, L. Velasco, "Machine Learning for Network Automation: Overview, Architecture, and Applications," in Journal of Optical Communications and Networking, vol. 10, issue 10, pp. D126-D143, 2018.

[5] ETSI GS NFV-IFA 027, "Network Functions Virtualisation (NFV) Release 2; Management and Orchestration; Performance Measurements Specification," 2018.

[6] E. Chuah et al., "Linking Resource Usage Anomalies with System Failures from Cluster Log Data," 2013 IEEE 32nd International Symposium on Reliable Distributed Systems, pp. 111-120, 2013.

[7] J. Hochenbaum, O. S. Vallis, A. Kejariwal, "Automatic Anomaly Detection in the Cloud via Statistical Learning," arXiv:1704.07706v1, 2017.

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

[9] J. Chen, M. Chen, X. Wei, and B. Chen, "Matrix Differential Decomposition-Based Anomaly Detection and Localization in NFV Networks," in IEEE Access, vol. 7, pp. 29320-29331, 2019.

[10] C. Sauvanaud, K. Lazri, M. Kaaniche, K. Kanoun, "Anomaly Detection and Root Cause Localization in Virtual Network Functions," 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), pp. 196-206, 2016.

[11] Y. Tan, H. Nguyen, Z. Shen, X. Gu, "PREPARE: Predictive Performance Anomaly Prevention for Virtualized Cloud Systems," 2012 IEEE 32nd International Conference on Distributed Computing Systems, pp. 285-294, 2012.

[12] J. Qiu et al., "Performance Anomaly Detection Models of Virtual Machines for Network Function Virtualization Infrastructure with Machine Learning," 2018 27th International Conference on Artificial Neural Networks (ICANN), LNCS 11140, pp. 479-488, 2018.

[13] H. Bouattour, Y. B. Slimen, M. Mechteri, H. Biallach, "Root Cause Analysis of Noisy Neighbors in a Virtualized Infrastructure," 2020 IEEE Wireless Communications and Networking Conference (WCNC), pp. 1-6, 2020.

[14] J. A. Cid-Fuentes, C. Szabo, K. Falkner, "Adaptive Performance Anomaly Detection in Distributed Systems Using Online SVMs," in IEEE Transactions on Dependable and Secure Computing, pp. 1-14, Apr. 2018.

[15] Grid Resource Allocation Agreement Protocol Working Group, "Web Services Agreement Specification (WS-Agreement)," [Online]. Available at https://www.ogf.org/documents/GFD.192.pdf.

[16] M. Guillame-Bert and O. Teytaud, "Exact distributed training: Random forest with billions of examples," in arXiv preprint, abs/1804.06755, 2018.

[17] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," in Frontiers in neurorobotics, vol. 7, pp. 1-21, Dec. 2013.

[18] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794, 2016.

[19] J. Schmidhuber, "Deep learning in neural networks: an overview," in Neural Networks, vol. 61, pp. 85-117, 2015.

[20] DPNM, "Network Intelligence Project," [Online]. Available: https://github.com/dpnm-ni.

[21] "Stress-ng - a tool to load and stress a computer system," [Online]. Available at https://manpages.ubuntu.com/manpages/artful/man1/stress-ng.1.html.

[22] "tc - show/manipulate traffic control settings," [Online]. Available at http://man7.org/linux/man-pages/man8/tc.8.html.

[23] "collectd - the system statistics collection daemon," [Online]. Available at https://collectd.org/.

[24] S. N. Z. Naqvi, S. Yfantidou, E. Zimányi, "Time series database and influxdb," Studienarbeit, Université Libre de Bruxelles, 2017.

[25] "Grafana," [Online]. Available at https://grafana.com/.

[26] The H2O.ai team, "H2O: Scalable Machine Learning," [Online]. Available at http://www.h2o.ai.

[27] The netfilter.org project, "iptables," [Online]. Available at http://ipset.netfilter.org/iptables.man.html.

[28] OISF, "Suricata - Open source IDS/IPS/NSM engine," [Online]. Available at https://suricata-ids.org/.

[29] The ntop team, "ntopng - High-Speed Web-based Traffic Analysis and Flow Collection," [Online]. Available at https://www.ntop.org/products/traffic-analysis/ntop/.

[30] L. Deri, M. Martinelli, T. Bujlow and A. Cardigliano, "nDPI: Open-source high-speed deep packet inspection," 2014 International Wireless Communications and Mobile Computing Conference (IWCMC), Nicosia, pp. 617-622, 2014.

[31] "HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer," [Online]. Available at http://www.haproxy.org/.