Open access • Proceedings Article • DOI:10.1109/DSN-W.2016.17

Towards Black-Box Anomaly Detection in Virtual Network Functions

— Source link

Carla Sauvanaud, Kahina Lazri, Mohamed Kaaniche, Karama Kanoun

Published on: 28 Jun 2016 - Dependable Systems and Networks

Topics: Virtual network, Virtualization, Service provider, Hardware virtualization and Network service

Towards Black-Box Anomaly Detection in Virtual Network Functions

Carla Sauvanaud, Kahina Lazri, Mohamed Kaâniche, Karama Kanoun

Towards Black-Box Anomaly Detection in Virtual Network Functions

Carla Sauvanaud*, Kahina Lazri†, Mohamed Kaâniche*, Karama Kanoun*
*LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
†Orange Labs, 38 rue du General Leclerc, 92130 Issy-Les-Moulineaux, France
Emails: firstname.name@laas.fr, firstname.name@orange.com

***Abstract*—The maturity of hardware virtualization has motivated communication service providers to apply this paradigm to network services. Virtual Network Functions (VNFs) come from this motivation and refer to any virtual execution environment configured to provide a given network service. VNFs constitute a new paradigm and related dependability evaluation mechanisms are still not thoroughly defined. In this paper we propose a preliminary evaluation of an anomaly detection approach applied to VNFs. Our approach uses a supervised machine learning algorithm. It notably relies on data provided by the underlying hypervisor of the VMs hosting the VNF, making it a black-box approach. Such an approach is actually well suited for infrastructure or telecommunication service providers willing to deploy tools that are easily configurable while reducing deployment costs. We validate our approach with the case study of the vIMS (IP Multimedia Subsystem) implemented by the Clearwater project.**

Keywords*—*VNF, fault injection, machine learning, black-box.

I. Introduction

Network functions (NFs) were for a long time inseparable of specialized proprietary hardware and stringent topology requirements. With the maturity of virtualization technologies, communication service providers led multiple initiatives to extend hardware virtualization to network services. These initiatives gave rise to the Network Function Virtualization paradigm (NFV). Moreover, as cloud computing emerged from virtualized infrastructures, it became attractive to host NFs as virtual machines running on virtualized and well orchestrated COTS (Commercial Off The Shelf) servers. The objective of NFV is to integrate network and IT domains in the same virtualized datacenters and to consolidate their separated management layers into a single logically centralized entity. NFV promises operating expenditure (OPEX) savings, quick deployments, scalability and high flexibility in the management of virtualized NFs.

The ETSI Industry Specification Group (ETSI-ISG) [1] defines the architecture of NFV by means of three functional entities, i) Virtual Network Functions (VNFs): any virtual execution environment configured to provide a given network service, ii) NFV infrastructure (NFVI): provides the execution environment for the VNFs, including the hardware resources and the virtualization layer which abstracts hardware resources, iii) NFV Management and Orchestration (NFV MANO): comprises the layers notably responsible for the management of the NFVI and the life cycle of the instantiated VNFs.

While several NFV use cases are already defined by the ETSI-ISG [1], some major aspects of this technology are still fuzzy. In particular, performances of NFs when migrated to cloud datacenters are one of the important challenges that needs to be addressed before the global adoption of NFVs. Indeed, NFs are more performance stringent than common IT applications. Nonetheless, as a result of their softwarization they are exposed to failures already present in these applications, and also to categories of failures related to virtualized infrastructures. For instance, virtualization overhead, issues in resource sharing, and hypervisor or VM manager scheduling failures are threats that should be handled by NFV.

Most of recent research work is focused on the implementation of single VNF like virtualized Deep Packet Inspection functions (vDPI), offered as a cloud service [2] or Software Defined Monitoring functions by Choi et al. [3]. Performance evaluation has also been subject of an active research effort. Performance characterization of VNFs is considered in NFV VITAL [4] and the evaluation of the NFVI infrastructure itself is addressed in the work of Cotroneo et al. [5]. NFV is also considered by equipment vendors which implement large NFV deployments like HP OpenNFV [6] with an implementation of large NFV deployment composed of several NFV components, Alcatel-Lucent Cloudband [7], Intel Open Network Platform [8], or CloudNFV [9]. Moreover, recent research results demonstrate the sensitivity of NFV to IT failures. In [10], Ge et al. show that NFV on COTS servers may need hardware acceleration to handle network processing tasks such as Deep Packet Inspection, Network Deduplication, or Network Address Translation. In adition, some virtualization features like live VM migration may also represent a source of performance degradation for VNFs since it may cause unacceptable downtime [11].

The objective of this paper is to evaluate the efficiency of a black-box anomaly detection approach applied on a VNF and relying on monitoring data sourced from the underlying hypervisor. A black-box approach is indeed well suited for infrastructure or telecommunication service providers willing to deploy tools that are easily configurable while reducing deployment costs. As a first step towards this objective, a set of fault campaigns are firstly defined and injected into a VNF execution environment in order to evaluate what impact they may have under operational conditions. In a second step, an anomaly detection approach is applied, on a per-VM basis, to detect any deviation of the VNF VMs from their expected behavior. One of the main asset of the approach is to ease automatic root cause analysis in that it can locate the potential faulty VM on which the anomaly is happening. Our experiments are performed with a real case study, namely a vIMS (IP

Multimedia Subsystem) [12] implemented by the Clearwater project. This vIMS implements multiple components, chained together dynamically to create a VNF-Forwarding Graph.

In the following, we first describe our testbed in section II. Section III presents our anomaly detection approach, and section IV provides preliminary results of our study. Finally, we conclude and discuss about future work in section V.

II. Case study and experimental platform

Our deployment is composed of three main entities: the Clearwater VNF, the IMS workload generated by the SIP benchmark, and the monitoring module. These entities are detailed below, after a brief description of the NFVI platform.

A. NFVI testbed

The virtualized platform is a VMware vSphere 5.1 private platform composed of 2 servers Dell Inc. PowerEdge R620 with Intel Xeon CPU E5-2660 2.20GHz and 64GB memory. Each server has a VMFS storage. One hypervisor hosts the benchmark and the other hosts all Clearwater components VMs. Each VM has 2 CPUs, 10GB memory, a 10GB disk. VMs are connected through a 100Mbps network.

B. Clearwater

Clearwater is an open source implementation of IMS for the cloud. It provides SIP-based (Session Initiation Protocol) voice and video calling, and messaging applications. It implements key standardized interfaces and functions of an IMS (except a core network) which enable industries to easily deploy, integrate and scale an IMS. Clearwater was initially designed for COTS servers of clouds. It is consequently well suited for NFV related studies. It encompasses six software components, namely Bono, Sprout, Homestead, Homer, Ralf, and Ellis. They are described below and represented in figure 1.

Bono is the SIP proxy in charge of implementing the P-CSCF function (Proxy-Call/Session Control Functions). It handles the users requests and routes them to Sprout. It also performs Network Address Translation traversal mechanisms.

Sprout is the IMS SIP router receiving requests from Bono and routing them to the adequate endpoints. It implements some S-CSCF (Serving-CSCF) and I-CSCF (Interrogating-CSCF) functions and gets the required users profiles and authentication data from Homestead. Sprout can also call application servers and actually contains itself a multimedia telephony (MMTel) application server, whose data are stored in homer (when calls are configured to use its services).

Homestead is an HTTP RESTful server and stores Home Subscriber Server data in a Cassandra database (i.e. information about subscribed services and locations). It is in charge of some I-CSCF and S-CSCF functions.

Thus, Bono, Sprout, and Homestead work together for the access control initiated by users and handle the entire CSCF.

Homer is a XDMS (XML Document Management Server) server with an XML Configuration Access Protocol Server (XCAP) interface, and runs a Cassandra database. It stores configuration information about MMTel service.

Ralf is the CTF (Charging Trigger Function). It bills the events collected by Bono and Sprout and reports them to a Charging Data Function server.

Ellis is a provisioning portal offering a web interface to users for testing purposes.

The IMS scales-out horizontally by means of a simple DNS load balancing mechanism. Our testbed encompasses Bono, Sprout, Homestead and Homer, each one deployed on one VM (see figure 1). As for preliminary results, the billing function is not configured, so Ralf is not included in our testbed. Neither is the testing component Ellis, and our deployment does not encompass redundancy.

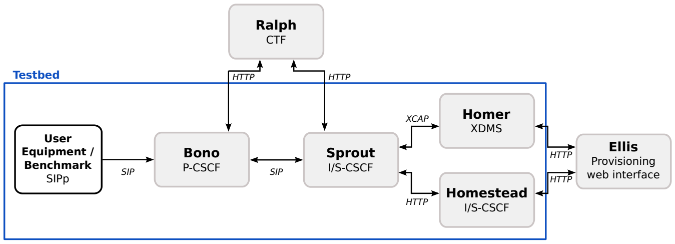

Fig. 1. Testbed deployment and Clearwater components.

C. Benchmark

IMS workloads are emulated by means of the SIPp benchmark[1]. The benchmark is configured with a fixed number of calls per second (we call it *load rate*) to be sent to the IMS. Each workload is defined by a scenario of SIP transactions in XML. One transaction corresponds to one message to be sent and an expected response message. The scenario simulates a standard call between two users and encompasses the standard SIP REGISTER, INVITE, UPDATE, and BYE messages.

The benchmark executes as many instances of scenario (i.e. *calls*) so as to comply with the load rate parameter. A call fails when a transaction fails. A transaction fails when a message is not sent or when an expected message is not received because of a timeout or an HTTP error code is received.

D. System observation

Computing and communication systems behaviors can be studied by means such as the monitoring of performance metrics, the analysis of audit data, or the analysis of OS or application logs. This paper specially focuses on the monitoring of performance data (such as CPU consumption, disk I/O, free memory...) of our Clearwater reference system. These data are collected through the VNF hypervisor API following a black-box approach. In other words, our approach does not require knowledge about the VNF implementation, and it can easily be applied to all types of VNF.

A vector of metrics collected at a given timestamp corresponds to a monitoring observation (also referred to as *observation*). Observations are collected from the VNF VMs every 15sec. A monitoring entity centralizes all VMs observations in a database. Nonetheless, the observations of each VM are studied separately in order to perform anomaly detection in each single VM. We use Pysphere[2] that communicates with

[1] http://sipp.sourceforge.net/index.html

[2] https://pypi.python.org/pypi/pysphere

the VMware SDK to collect metrics from the VMs. Each observation is composed of the 152 metrics listed online[3].

III. Approach and assessment

A. Anomaly detection approach based on machine learning

In previous work [13], [14] we defined an approach based on machine learning for anomaly detection in virtualized infrastructures (we tested our approach on testbeds hosting MongoDB and VoltDB database). In this paper, we evaluate to what extent such an approach is efficient to the detection of anomalies in VNFs. We base our work on the application of a supervised learning algorithm named Random Forest, to detect anomalies based on monitoring data. This algorithm is well known to provide good classification results and to handle a large number of vectors features (i.e. observations metrics).

In order to apply a supervised learning, we need to provided the algorithm with training data that encompass data representing normal behaviors but also data representing abnormal behaviors so as to create a classifier. As a result, we use fault injection techniques during a training-purposed runtime of our VNF so as to deterministically trigger abnormal behaviors. In this way we obtain different classes of behaviors observations including anomalies in a short testing time.

Considering our testbed, anomaly detection is performed for each VM of the VNF components. Thus we get several classifiers that detect anomalies in single VMs. This method eases the root cause analysis by locating the potential faulty VM. Also, it is useful for the analysis of fault propagation between components of the same VNF.

B. Fault injection

Our injection scripts enable us to emulate five types of anomalies, namely 1) CPU consumption, 2) misuse of memory (i.e. memory increase), 3) anomalous number of disk access (i.e. increase of disk I/O access and synchronisations), 4) network packet loss, and 5) network latency. They are respectively referred to as CPU, memory, disk, packet loss, and latency injections.

In the following, we give further details about the emulated anomalies, the intensity levels of injections, their implementation and we describe the injection campaigns that we carried out.

1) Types of anomalies: **CPU consumption.** Anomalous CPU consumptions may arise from faulty programs encountering impossible termination conditions leading to infinite loops, busy waits or deadlocks of competing actions, which are common issues in multiprocessing and distributed systems.

Memory leaks. Anomalous memory usages are common whose allocated chunks of memory are not freed after their use. Accumulations of unfreed memory may lead to memory shortage and system failures. We believe that such cloud-related mechanism as ballooning may also lead to such failures.

Anomalous number of disk access. A high number of disk accesses, or an increase of disk accesses over a short period of time, emulates anomalous disks whose accesses often fail and lead to an increase in disk access retries. It may also result from a faulty program stuck in an infinite loop of data writing.

Network anomaly. Such anomalies may arise from network interfaces or the interconnection of networks. We emulate packet losses, and latency increases. Packet losses may arise from undersized buffers, wrong routing policies and even firewall misconfigurations. Latency anomalies may originate from queuing or processing delays of packets on gateways.

2) Intensity levels: Each anomaly has two intensity levels, namely medium and high, that we calibrated based on prior experimentations on the platform. Regarding the memory, disk and CPU anomalies, the maximum intensity value of an anomaly is constrained by the capacity of VMs operating systems. In other words, the high intensity injection is the maximum resource consumption that is tolerated by the OS before it kills the injection script or triggers protection mechanisms that prevent the injection program to consume more resources. Considering the remaining types of injections, the maximum intensity value is set so as not to lead to a maximum SLA violation rate, i.e. 100% failed transactions but to be close to it (around 99%). Intensity levels are described in Table I.

TABLE I. Anomalies types and intensities.

Type	Unit	Intensity levels	
		Medium	High
CPU consumption	%	70	97
Misuse of memory	%	70	97
Disk access	#workers	40	60
Packet loss	%	5.0	10.0
Network latency	ms.	50	100

3) Injection campaigns implementation: Injections are performed by scripts stored on each VMs disks. They are run through an SSH connexion orchestrated by a campaign handler script which is a configurable script hosted on the benchmark VM. We use the Linux kernels tools *iptables* and *tc* for the injection of network latencies on the POSROUTING chain, and *iptables* on the INPUT and OUTPUT chains for the injection of packet losses. The stress test tool Stress-ng[4] is run for CPU, disk, and memory related injections.

Injection campaigns target Sprout, Bono, and Homestead, which implement the CSCF of Clearwater. In each campaign, the benchmark is run for 80min. The load rate of the benchmark is calibrated to lead to less than a 1% failure rate of the IMS without injections (failures might arise from a load being too high because of bottlenecks in Sprout **??**). After 40min an anomaly of 3min is injected. The injection time is calibrated to affect several instances of scenario executions (an execution lasts surely less than 1sec) and not to be too short regarding our monitoring period of 15sec. Thus, each campaign contains one injection of one intensity in one target VM. Finally, each campaign ends up with the reboot of all the VMs of the testbed.

IV. Results

We present a few examples of experimental results illustrating the effectiveness of our approach to detect injection-related abnormal behaviors, by computing the Receiver Operating Characteristic (ROC) (i.e. the true positive rate against the

[3] https://homepages.laas.fr/csauvana/datasets/pysphere_vm_counters.txt

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

false positive rate) and the Precision-Recall (PR) curve (i.e. precision-recall pairs for different classification probability thresholds). For both curves, a perfect classifier would have an Area Under the Curve (AUC) of 1.

As we ran the injection campaigns presented in III-B3, we collected around 9600 observations for each VM hosting a Clearwater component. We ran the Random Forest algorithm over these observations for a binary detection (class 0: no injection; 1: injection) and a multi-class detection with the training of one model for each class (class 0: packet loss injection; class 1: latency injection...). We first validated our datasets by cross validation and obtained a standrad deviation of the ROC and PR AUC of 0.02. The best detection results are obtained when the Random Forest algorithm is configured with 10 decision trees. ROC and PR curves for the detection from Sprout monitoring observations and for the detection of injections that were injected in Sprout are presented respectively in figure 2 and 3. We obtain similar results for the detection of injections in Bono (resp. Homestead) from Bono (resp. Homestead) observations.

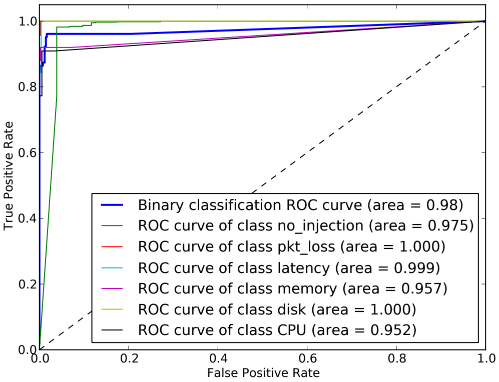

Fig. 2. ROC curves for multi-class and binary classification.

Firstly, the ROC and PR AUC for the binary classification case are respectively $0, 98$ and 0.94, which corresponds to high precision and recall with a low false alarm rate. In addition the ROC AUC for the multi-class detection varies between 1 and 0.95, and the PR AUC varies between 1 and 0.86. Thus, results are encouraging and make our approach applicable to real case scenario of online anomaly detection. In addition, the nature of injections can be indentified by the models and help root cause analysis. Regarding the PR curves, CPU injections are more difficult to discern from other injections and a normal behavior with a PR AUC of 0.86. The algorithm might confuse such injections with disk or memory injections which also use additional CPU time so as to process the injections scripts.

V. Conclusions and Future work

In this paper we evaluated the ability of the Random Forest algorithm to detect abnormal behaviors in the Clearwater VNF with a black-box approach. Our approach enables an excellent detection of anomaly injections on a per-VM basis, with an area under the PR and ROC curves from 0.87 up to 1 depending on the type of anomaly for a multi-class detection. It may also discern different classes of injections such as CPU exhaustion, memory shortage, excessive disk I/O, and network anomalies. Thus it actually does ease the root cause analysis and reconfiguration actions. As a future work, we plan to evaluate with the same approach the behaviors of Clearwater components during cloud management actions such as ballooning and live VM migration. In this way we will test whether and to what extent such actions might impact the IMS operation and whether the NFV MANO should make use of such techniques. We also intend to work on the impact of our injections on the IMS SLA.

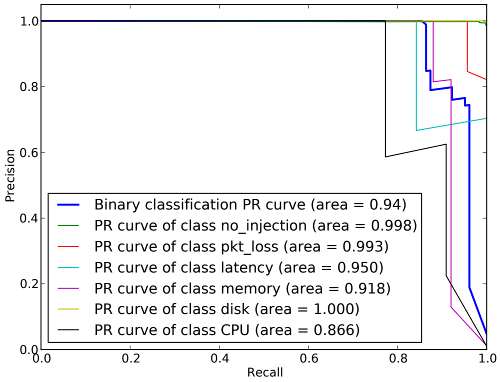

Fig. 3. PR curves for multi-class and binary classification.

References

[1] ETSI Group Speciafication NFV. (2013) Use cases 001 v1.1.1.

[2] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral, “Deep packet inspection as a service,” in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT ’14. New York, NY, USA: ACM, 2014, pp. 271–282.

[3] T. Choi, S. Kang, S. Yoon, S. Yang, S. Song, and H. Park, “Suvmf: Software-defined unified virtual monitoring function for sdn-based large-scale networks,” in *Proceedings of The Ninth International Conference on Future Internet Technologies*, ser. CFI ’14. New York, NY, USA: ACM, 2014, pp. 4:1–4:6.

[4] L. Cao, P. Sharma, S. Fahmy, and V. Saxena, “Nfv-vital: A framework for characterizing the performance of virtual network functions,” in *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on*, Nov 2015, pp. 93–99.

[5] D. Cotroneo, L. De Simone, A. Iannillo, A. Lanzaro, and R. Natella, “Dependability evaluation and benchmarking of network function virtualization infrastructures,” in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, April 2015, pp. 1–9.

[6] Hewlett Packard Enterprise. (2016) Hp opennfv reference architecture. http://www8.hp.com/us/en/cloud/nfv-overview.html?

[7] Alcatel-Lucent. (2016) Alcatel-Lucent ClouBand. http://www.alcatel-lucent.com/solutions/cloudband.

[8] Intel. (2016) Intel open network platform. http://www.intel.com/ONP.

[9] CloudNFV. (2016) Cloudnfv. http://www.cloudnfv.com/WhitePaper.pdf.

[10] X. Ge, Y. Liu, D. H. Du, L. Zhang, H. Guan, J. Chen, Y. Zhao, and X. Hu, “Openanfv: Accelerating network function virtualization with a consolidated framework in openstack,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 353–354, Aug. 2014.

[11] Openstack. (2016) Live Migration at HP Public Cloud. https://www.openstack.org/summit/vancouver-2015/summit-videos/presentation/live-migration-at-hp-public-cloud.

[12] Project Clearwater. (2016) http://www.projectclearwater.org/.

[13] G. Silvestre, C. Sauvanaud, M. Kaâniche, and K. Kanoun, “An anomaly detection approach for scale-out storage systems,” in *26th International Symposium on Computer Architecture and High Performance Computing*, Paris, France, Oct. 2014.

[14] ——, “Tejo: A Supervised Anomaly Detection Scheme for NewSQL Databases,” in *7th International Workshop on Software Engineering for Resilient Systems (SERENE 2015)*, Paris, France, Sep. 2015.