Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 110 (2017) 510–515

Procedia Computer Science

www.elsevier.com/locate/procedia

2nd International Workshop on Applications of Software-Defined Networking in Cloud Computing (SDNCC)

Automated Anomaly Detection in Virtualized Services Using Deep Packet Inspection

Marcel Wallschläger[a], Anton Gulenko[a], Florian Schmidt[a], Odej Kao[a], Feng Liu[b]

[a]Technische Universität Berlin (TU Berlin, Complex and Distributed IT Systems (CIT)), 10587 Berlin, Germany
[b]Huawei European Research Center, Huawei Technologies Co., Ltd., 80992 Munich, Germany

Abstract

Virtualization technologies have proven to be important drivers for the fast and cost-efficient development and deployment of services. While the benefits are tremendous, there are many challenges to be faced when developing or porting services to virtualized infrastructure. Especially critical applications like Virtualized Network Functions must meet high requirements in terms of reliability and resilience. An important tool when meeting such requirements is detecting anomalous system components and recovering the anomaly before it turns into a fault and subsequently into a failure visible to the client.

Anomaly detection for virtualized services relies on collecting system metrics that represent the normal operation state of every component and allow the usage of machine learning algorithms to automatically build models representing such state. This paper presents an approach for collecting service-layer metrics while treating services as black-boxes. This allows service providers to implement anomaly detection on the application layer without the need to modify third-party software. Deep Packet Inspection is used to analyse the traffic of virtual machines on the hypervisor layer, producing both generic and protocol-specific communication metrics. An evaluation shows that the resulting metrics represent the normal operation state of an example Virtualized Network Function and are therefore a valuable contribution to automatic anomaly detection in virtualized services.

Keywords: Cloud, Network, Deep Packet Inspection, Network Function Virtualization, Anomaly Detection

1. Introduction

As the number of services running on virtualized infrastructure continues to increase, even critical applications from the telecommunication sector are ported from traditional appliances onto cloud deployments.

Driven by high customer expectations, such critical services have an especially high demand for reliability and continuous service delivery.

* Marcel Wallschläger. Tel.: +49-30-314-78592 ; fax: +0-000-000-0000 .
E-mail address: marcel.wallschlaeger@tu-berlin.de

1877-0509 © 2017 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the Conference Program Chairs.
10.1016/j.procs.2017.06.137

Traditional reactive fault management is not sufficient to guarantee a constantly high service availability. Cost effective private and public clouds rely on commodity hardware which cannot guarantee failover latencies short enough to hide faults from clients.

Since anomalies often precede faults, anomaly detection mechanisms provide an early warning system that enables proactive fault management[1]. Anomaly detection can operate across all layers of the cloud system by collecting various operative time series metrics. Unsupervised machine learning techniques analyse the collected data to detect abnormal patterns and outliers.

The anomaly detection accuracy mainly depends on the input data used to build the underlying machine learning models. In the context of virtualized services, many indicators for normal operation can be found in resource usage data like the utilization of CPU, memory, network and disk. This data is available on all system layers and does not require specific knowledge about the monitored service. Treating services as black boxes makes this anomaly detection approach easily usable in arbitrary productive environments. Furthermore, it allows Infrastructure-as-a-Service (IaaS) providers to offer an anomaly detection service to customers while maintaining minimal interference with customer machines.

However, mere resource usage fails to accurately reflect the communication patterns of the observed services and therefore does not cover all potential anomalies. This paper presents a mechanism for collecting service communication metrics while still remaining largely service agnostic, as long as the services rely on standardized application layer protocols. Deep Packet Inspection (DPI) on the hypervisor level allows real-time analysis of the inter-service communication with low interference. Services do not need to be customized or extended to obtain this information, but knowledge about the used protocols can reduce the performance overhead of packet inspection. The second contribution of this paper is an evaluation of the presented data collection mechanism. We show that a number of simulated anomalies manifest themselves in the resulting metrics.

The remainder of the paper is organized as follows. The following section presents related research in the field of deep package inspection and anomaly detection. Section 3 describes the presented approach in detail, while section 4 presents an evaluation thereof.

2. Related Work

Much related work has been conducted in the field of deep package inspection and automated protocol analysis.

M. Danelutto et al. show in their work[2] how package inspection is possible on commodity server hardware using a skeleton-based parallel programming library targeting efficient streaming on multi-core architectures. Using their framework the authors show that package inspection for packets with 60 byte payloads can be performed with a commodity Intel 10 Gbit network card.

Anat Bremler-Barr et al. identify DPI as a common task for middleboxes that inspect application layer packets many times during their route from sender to the final destination[3]. Since most middleboxes perform similar analyses to implement traffic control and QoS measurements, the authors propose a DPI-as-a-service infrastructure to reduce the total overhead of repeated packet inspection. The proposed infrastructure can lead to improved performance, scalability and robustness, showing that DPI has become mature and fast enough to be executed even as a standalone service.

Further research shows that the application layer protocol used by packets can be identified in real-time using DPI[4,5,6].

3. Approach

An exemplary infrastructure illustrates the proposed data collection approach and follows a typical cloud deployment used both by IaaS providers and in private clouds hosting Virtualized Network Function services. The services run on virtualized hardware and network resources provided by the cloud operating system OpenStack[7] and complementary Software Defined Networking (SDN) components. Open vSwitch[8] manages the virtual network connecting all VMs.

Figure 1 shows the basic structure of the network architecture inside a physical compute node managed by OpenStack. It illustrates two virtual machines within one hypervisor connected to various Open vSwitch bridges. A dedicated monitoring agent implements the data collection by sniffing packets on the tap interface of every relevant virtual machine. The tap interface transfers every packet received and sent by the virtual machine, after being filtered by the Security Groups mechanism of OpenStack. Security Groups implement a simple rule-based firewall managed by the owner of the virtual machine. Malicious or unintended packets are dropped at this point, resulting in increased anomaly detection precision when analysing packets on the tap interface. Detecting malicious activities is not the focus of application-layer anomaly detection, and is left for dedicated Intrusion Detection Systems (IDS).

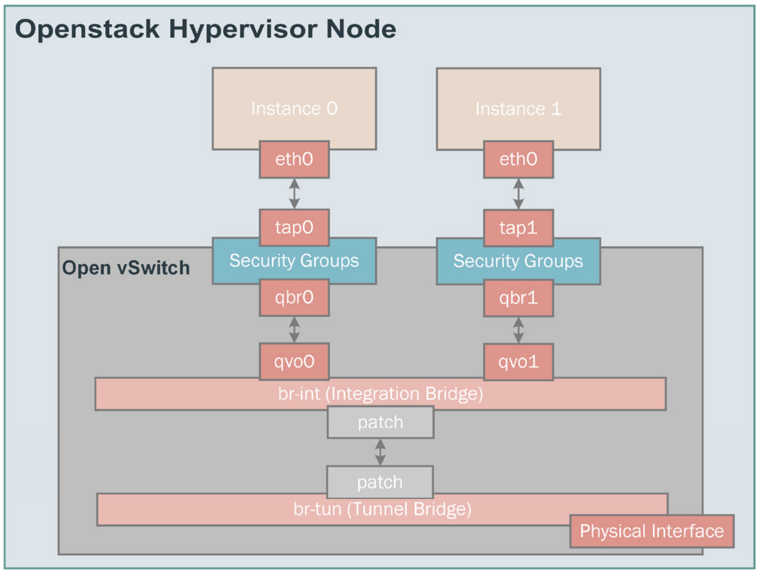

Figure 1: Hypervisor Structure

The data collection agent analyses the observed traffic and extracts metrics that are specific to different Layer 7 protocols. The used protocols can be inferred in real-time[4,5,6], but the performance overhead can optionally be reduced when this information is provided by the user. This knowledge allows an additional optimization which is to only analyse traffic on specified ports that belong to relevant application layer protocols. In a well-configured OpenStack environment, such information can be obtained by analysing all security groups, since they explicitly whitelist all connections expected between service instances.

For every relevant application layer protocol, the data collection agent counts the occurrence of specific messages and reports how often each class of messages is observed in a configurable time interval. The message classes depend on the specific protocol and in order to maintain an appropriate number of metrics, some message classes have to be grouped into one. HTTP communication, for example, consists of requests and responses, where the response can contain one of dozens standardized status codes. Instead of counting every code individually, the codes are grouped by their first digit: 1xx, 2xx, 3xx, and so on. This results in 5 metrics that provide a good approximation of the operation of an HTTP server. A simliar grouping mechanism is used to track the Session Initiation Protocol (SIP)[9]. In addition to protocol specific metrics, the data collection agent provides generic information for every observed port, like the number of transmitted packets and bytes.

Table 1 gives an overview over the metrics obtained from the SIP protocol. The SIP-protocol is use commonly in IP-telephony such as VoIP. SIP controls the multimedia sessions and handles creation, termination and modification of media streams such as voice-connections. It is a text-based protocol using elements of Hypertext Transfer Protocol (HTTP). The data collection agent is written in Python using the scapy library[*]. The collected statistics are recorded every 500 milliseconds.

[*] https://github.com/phaethon/scapy

Table 1: abstracted features from SIP protocol handler

Feature Names	Description
Overall packets	Number of Packets
Registrations	Registration requests
Return code 1xx	Indicates the request was valid and is being processed
Return code 2xx	Indicates successful completion
Return code 3xx	Indicates a redirection is needed
Return code 4xx	Indicates bad syntax or cannot be fulfilled
Return code 5xx	Indicates server failure in processing request
Return code 6xx	Indicates global failure

4. Evaluation

An evaluation inside a dedicated private cloud testbed demonstrates that a selected set of anomalies can be observed in the metrics produced by the proposed data collection approach. The testbed is based on OpenStack Kilo and runs Open vSwitch in version 2.41. The setup simulates an IaaS provider hosting customer services on five hypervisors and one OpenStack controller node, which is also running the OpenStack networking service.

The client service is Project Clearwater[10], an open source implementation of the IP Multimedia Subsystem (IMS). An IMS service uses the SIP protocol to establish sessions between clients. Figure 2 shows the architecture of Project Clearwater. The red box highlights the SIP communication channel between the two core components called Bono and Sprout. The data collection agent monitors the SIP communication on this connection. The Bono node acts as a client-facing SIP proxy, while Sprout implements the core IMS functionality. To fulfill its role, Sprout needs a working connection to Homestead to obtain information about registered clients.

The SIP benchmarking client sip-stress[‡] continuously exchanges SIP messages with the Bono node, which also leads to traffic on the Sprout and Homestead components.

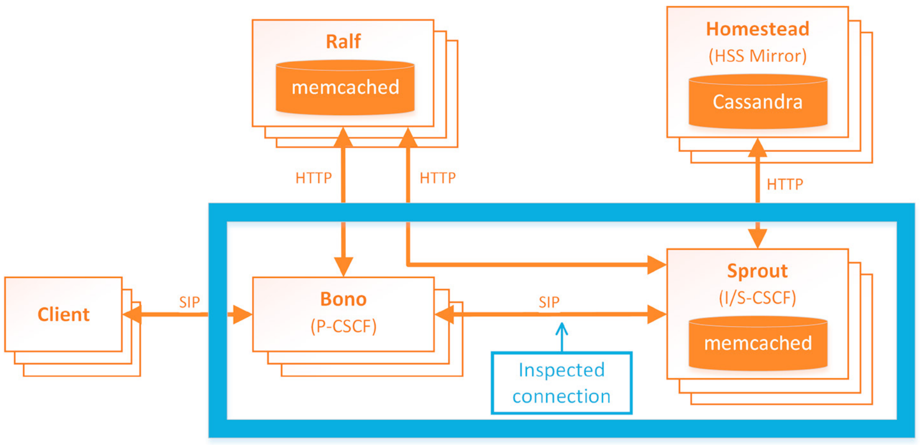

Figure 2: Excerpt of the Project Clearwater Architecture including connection protocols

[‡] http://clearwater.readthedocs.io/en/stable/Clearwater_stress_testing.html

Two anomalies are injected into the running system to test the reaction of the collected metrics:

1. Increasing the network latency of the virtualized NIC of the Sprout VM by 150ms. This anomaly represents a degraded state of a network component which does not result in system crashes but reduces the performance of the system.
2. Stopping the Homestead service running on a separate virtual machine. This anomaly affects the entire Clearwater installation, since client registrations are not possible anymore. Therefore, the anomaly propagates to the Sprout and Bono component, and finally to the client of the system.

One evaluation experiment consists of two steps. First, the data collection agent records data during normal system operation for one minute. Afterwards, one of the above anomalies is injected and the system state is observed for another minute, before reverting the anomaly. This procedure is repeated ten times.

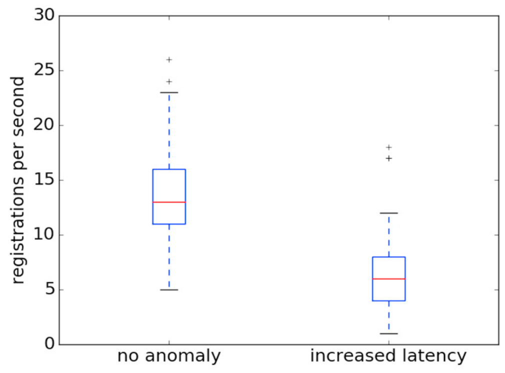

Figure 3a: Registrations per second, increased latency

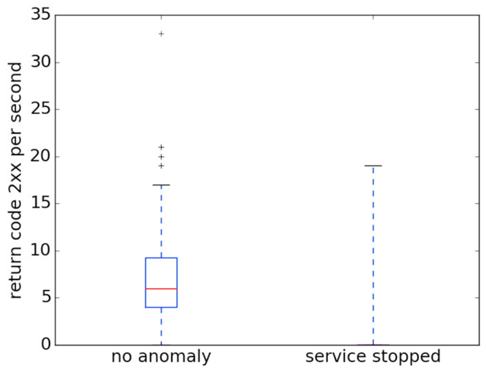

Figure 3b: Success messages per second, Homestead service stopped

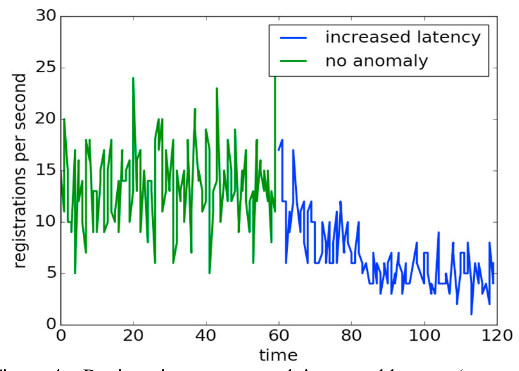

Figure 4a: Registrations per second, increased latency (one run)

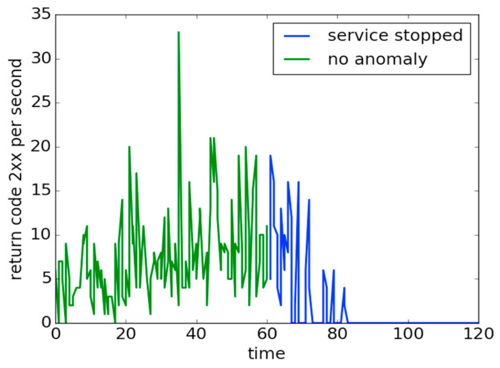

Figure 4b: Success messages per second, Homestead service stopped (one run)

Figure 3a shows two box plots summarizing all measurements of the number of registrations per second with and without the increased latency anomaly. The amount of registrations per second is reduced while the anomaly is active. Figure 4a shows the development of the registrations per second metric within one exemplary evaluation run.

Figure 3b shows the number of success return messages during normal operation and after stopping the Homestead service. Since the Clearwater is unable to correctly process client authentication requests, the number of success messages rapidly drops to zero. Figure 4b shows the successful return messages during one run of the experiment.

This evaluation suggests that the selected anomalies are manifested in the metrics produced by the proposed data collection technique. The resulting metrics are therefore valuable input features for anomaly detection mechanisms in practice.

5. Conclusion

This work presents an approach for collecting metrics about the operation of virtualized services with minimal interference. This allows IaaS service providers to offer anomaly detection as a service to their clients, as well as generic anomaly detection mechanisms in private clouds. The presented approach is evaluated by injecting anomalies in a dedicated cloud testbed.

Future work on this topic includes further evaluations of DPI based metric collection in combination with online anomaly detection approaches. Handling more application layer protocols will extend the approach to a wider range of virtualized services. An extended evaluation in a fully loaded 10Gbit network on commodity hardware will further support the practical application of the presented approach.

References

1. Gulenko, Anton, et al. Evaluating machine learning algorithms for anomaly detection in clouds. In: *IEEE International Conference on Big Data (Big Data)*. 2016, pp. 2716-2721.
2. Danelutto, M., et al. Deep packet inspection on commodity hardware using fastflow. In: *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*. 2016, pp. 92-99.
3. Bremler-Barr, Anat, et al. Deep packet inspection as a service. In: *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 2014, pp. 271-282.
4. Riccobene, Vincenzo, et al. Automated generation of VNF deployment rules using infrastructure affinity characterization. In: *NetSoft Conference and Workshops (NetSoft)*. IEEE, 2016, pp. 226-233.
5. Hara, Masaki, et al. Service Identification by Packet Inspection Based on N-grams in Multiple Connections. In: *Fourth International Symposium on Computing and Networking (CANDAR)*. IEEE, 2016, pp. 686-690.
6. Husák, M., et al. A survey of methods for encrypted traffic classification and analysis. In: *International Journal of Network Management, Vol. 25*. 2015, pp. 355-374.
7. Openstack Foundation, 2017. URL: https://www.openstack.org.
8. Linux Foundation Collaborative Project, 2016. URL: http://openvswitch.org.
9. Rosenberg, J., et al. SIP: Session Initiation Protocol: RFC Editor, 2002.
10. Metaswitch Networks: Project Clearwater, 2016. URL: www.projectclearwater.org.