Network Intrusion Detection System as a Service in OpenStack Cloud

Chen Xu, Ruipeng Zhang, Mengjun Xie, Li Yang
Department of Computer Science and Engineering
The University of Tennessee at Chattanooga
{kjx384,smj793}@mocs.utc.edu,{Mengjun-Xie,Li-Yang}@utc.edu

***Abstract*—Network intrusion detection system (NIDS) is indispensable for cloud computing providers to detect ongoing cyber attacks and deter future ones in the cloud computing era. To help cloud users to secure their tenant networks inside a computing cloud, a tenant-based NIDS service model has been proposed where NIDS services are deployed as virtual instances inside tenants. However, this approach imposes significant virtualization overhead at the cost of tenants, and the NIDS provision process can be time-consuming. In this work, we present an innovative service model for OpenStack clouds called Network Intrusion Detection System as a Service (NIDSaaS). NIDSaaS enables on-demand, quick deployment and termination of NIDS, while maintaining lightweight overhead for cloud hosts and tenants. In addition, NIDSaaS provides a straightforward command line interface (CLI) so that cloud administrators can easily integrate NIDS with their tenants. We have implemented a prototype of NIDSaaS and evaluated it on a multi-node OpenStack testbed. Our evaluation results show that NIDSaaS outperforms existing VM-based NIDS service approach substantially in terms of service launch time and resource usage.**

***Index Terms*—Cloud Security, Neutron Plugin, Network Intrusion Detection System (NIDS), OpenStack**

I. Introduction

Self-hosted cloud solutions such as OpenStack, Eucalyptus, and OpenNebula have been extensively developed as cloud computing becomes increasingly popular. They provide multiple services to meet cloud computing's needs in computing, storage, and networking. However, those solutions often lack network intrusion detection function that can protect tenants and detect cyber attacks such as distributed denial-of-service (DDoS) attacks. Moreover, according to 2019 Verizon data breach investigations report [1], 34% of attacks involved insiders. Inside attackers can exploit cloud networking vulnerabilities against other tenants by port scanning, traffic sniffing, and traffic spoofing, and so on [2]. To protect cloud tenants from both external and internal attacks, cloud administrators can apply network intrusion detection systems (NIDSes) such as Snort and Suricata to detect malicious traffic and update firewall rules accordingly to block attack attempts.

Existing approaches to tenant network traffic monitoring and intrusion detection in a cloud leverage software-defined network (SDN) [3] and virtual machine (VM) technologies. Traditionally, the Switched Port Analyzer (SPAN, or port mirroring) [4] of a physical switch is used to duplicate network packets and forward these duplicated packets to a preset NIDS. In a virtualized environment, SPAN is supported by Open vSwitch (OVS) [5] for traffic mirroring, hence enabling the development of virtualized NIDS services in a cloud. However, VMs in the same tenant may be hosted by multiple hypervisors. This traffic mirroring method constrains the scope of VM monitoring to a single hypervisor. Traffic mirroring across multiple hypervisors can be achieved by an OpenStack plugin called Tap-as-a-Service (TaaS) [6], which provides port mirroring capability for tenant networks. Using TaaS, users can set up an NIDS service by launching an NIDS-enabled VM instance in the cloud and mirroring target network traffic to the instance for intrusion detection. However, this approach are costly due to the significant overhead incurred by VM invocation.

In this paper, we propose a novel NIDS service model for the OpenStack called NIDSaaS (NIDS as a service), which provisions on-demand NIDS services and provides suspicious traffic detection and analysis in tenant space. NIDSaaS is designed to dynamically create/destroy NIDS services in an OpenStack cloud and to update NIDS rule sets upon request in an efficient manner. Unlike VM-based NIDS deployment, we reduce the time of provisioning NIDS services and eliminate virtualization overhead by running NIDSaaS on designated host(s) directly. Since instances of a tenant may reside in multiple hosts, the traffic mirroring issue in using SDN and OpenFlow is also addressed in this work. We have developed an NIDSaaS prototype that consists of a user client, a service plugin, a service plugin agent, and a Snort-driven NIDS provider. We have evaluated NIDSaaS performance in terms of launch time, CPU and memory usage on a multi-node OpenStack platform. Our evaluation results show that NIDSaaS can achieve much shorter service launch time and substantially less CPU and memory consumption compared to the VM-based NIDS services through TaaS.

The rest of the paper is organized as follows. Section II briefly reviews the related work. Section III describes the NIDsaaS architecture, workflow and its network model in OVS-based OpenStack. Section IV evaluates the launch time and resource usage of NIDSaaS. Finally, Section V concludes this paper and discusses future work.

II. Related Work

There are two major intrusion detection techniques for NIDS: signature-based detection and anomaly-based detection [7]. Signature-based intrusion detection inspects network

978-1-7281-4905-9/20/$31.00 ©2020 IEEE

traffic and decides whether the extracted patterns of inbound network traffic match one or more predefined signatures (also called rules). Several popular NIDSes such as Snort [8] and Suricata [9] use this technology, which can attain a high level of accuracy for known attacks. However, signature-based NIDSes are not capable of detecting new, unknown intrusions. On the contrary, anomaly-based detection can detect previously unseen intrusion events by discovering deviation of anomalous network behavior from the normal one. Anomaly detection techniques can be classified into three main categories: statistical-based, knowledge-based, and machine learning-based [10]. Zeek [11] (previously named Bro) is capable of anomaly detection by implementing application-level semantics, event analysis, pattern matching, and protocol analysis.

One area of NIDS research focuses on making existing NIDSes more efficient and effective in computing clouds. In [12], the authors proposed a hierarchical and autonomous cloud-based intrusion detection system HA-CIDS, in which system events are first analyzed to compute the risk parameter and then the autonomous controller chooses appropriate response to threats. Meng *et al.* [13] designed a parallel model to conduct the exclusive signature matching in the cloud, which addresses the performance issue of signature-based IDS. Lin *et al.* [14] proposed an NIDS that can resolve the virtual system information from operating systems' kernel map in hypervisor layer. By doing so, services in a cloud can be identified and required detection rules can be tuned dynamically. In vNIDS [15], the authors addressed the challenges in effective intrusion detection and non-monolithic NIDS provisioning by applying detection state sharing and microservice techniques. The authors of Kitsune [16] utilized the neural network to track the patterns of network channels efficiently.

New NIDS models have been developed to cope with evolving network threats and provide new methods for intrusion detection. Carli *et al.* [17] studied how to handle large volumes of traffic and proposed a domain-specific concurrency model that can partition the network traffic and detect vulnerabilities. Kim *et al.* [18] proposed a hybrid intrusion detection method that integrates a misuse detection model and an anomaly detection model. They used a decision tree algorithm to justify the misuse behaviors and a single classification SVM algorithm for detecting network anomalies. K.Viegas *et al.* [19] proposed a new multi-objective feature selection method for network intrusion detection based on decision tree and naive Bayes classifiers, which improves detection accuracy by considering real-world network proprieties. A number of studies, e.g., [20]–[22], explored deep learning approaches to network intrusion detection. However, they often lack evaluation on real-time detection.

III. System Design

In this section, we present the architecture, workflow, and network model of NIDSaaS.

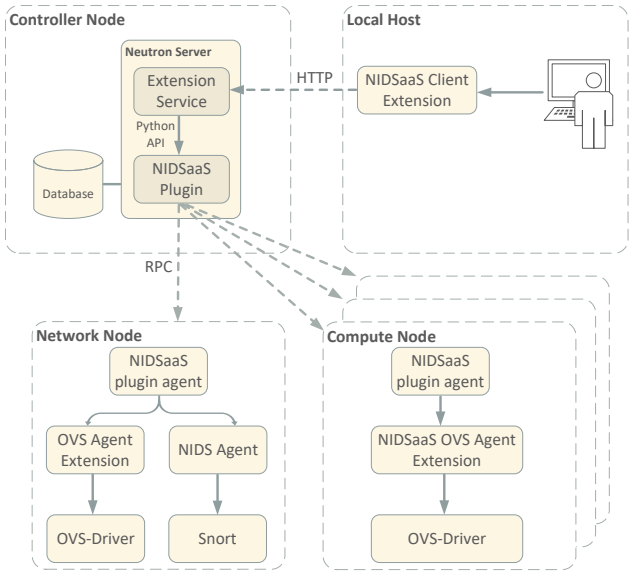


Fig. 1. Architecture of NIDSaaS

A. Architecture and Workflow

NIDSaaS is designed to provide fast and on-demand NIDS service provisioning to tenants. NIDSaaS operates across OpenStack controller, network, and compute nodes. It has three main components: NIDSaaS client extension, NIDSaaS plugin, and NIDSaaS plugin agent, as shown in Fig. 1. The NIDSaaS client extension extends the Neutron client and provides a suite of commands to help users manage their NIDS resources remotely. The NIDSaaS plugin loaded by the Neutron server deals with the user requests sent by the NIDSaaS client. It also communicates with NIDSaaS plugin agents via remote procedure call (RPC) to apply changes to the invoked NIDS services and their rule sets. NIDSaaS plugin agents are deployed on OpenStack network and compute nodes. They operate with Open vSwitch and OpenFlow to realize tenant traffic mirroring. They are also responsible for managing life cycle of NIDS services.

Fig. 2 shows how a user interacts with NIDSaaS and how NIDSaaS components interact with each other. There are three types of commands (denoted by ① ② ③) NIDSaaS exposes to a tenant administrator. Commands of type ① are used to create or destroy an NIDS service. Commands of type ② are for updating detection rules of a given NIDS service. Finally, by calling a command of type ③, users can mirror the network traffic from/to a specific Neutron port to the NIDS service, which eventually connects the NIDS service to the target tenant network. The traffic flow direction, source port, and destination port of the mirrored traffic define a NIDSaaS flow.

User requests triggered by these commands are sent to the Neutron server via HTTP REST (REpresentational State Transfer) APIs. The NIDSaaS plugin creates NIDSaaS APIs endpoints in the Neutron server, listens for client requests,

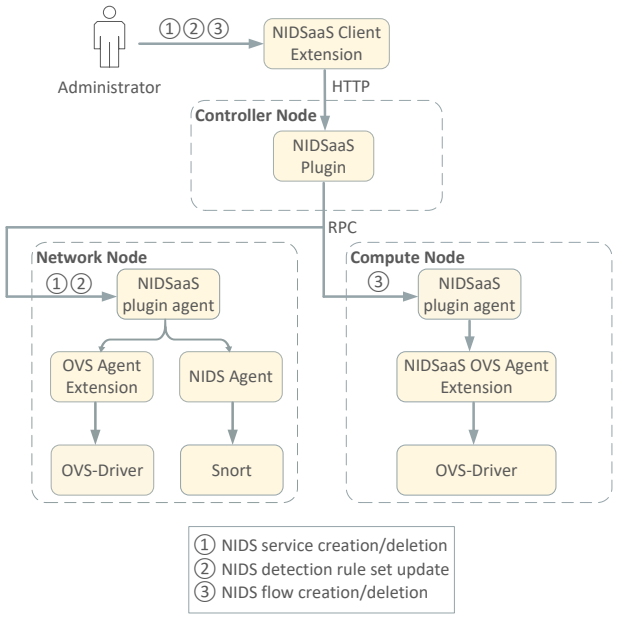


Fig. 2. Workflow of NIDSaaS

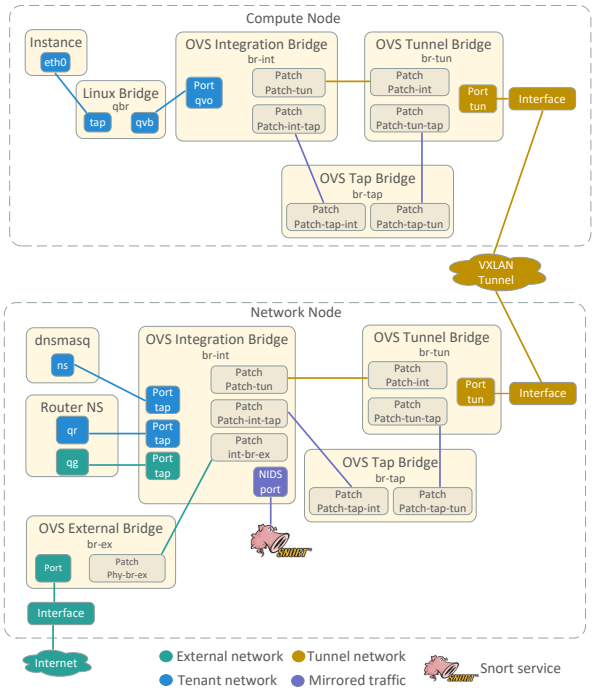


Fig. 3. Network Model

and invokes corresponding service plugin functions. For those requests from outside the cloud, they should be operated over TLS for security reason. In addition, the API services should be deployed behind a TLS termination proxy that can handle incoming TLS connections and decrypting the TLS and forwarding requests to the API services inside the cloud. Similar to other OpenStack service plugins, the NIDSaaS service plugin is responsible for storing necessary information into the database on the controller node such as network, subnet, tenant, NIDS port, and type of rule set, etc. Only cloud administrators can access or grant users to access the databases.

Upon receiving a command of type ①, the NIDSaaS plugin will send an RPC message to the NIDSaaS plugin agent on the network node. Besides creating a new NIDS service, this RPC message further triggers two essential tasks. First, the plugin agent creates a Neutron port (TAP port) for the NIDS service to listen on and also sets up OpenFlow rules for the OVS bridge so that it can deliver mirrored packets to the Neutron port. Second, the agent launches the NIDS service with the preset configuration and connects the service to the Neutron port. The plugin agent is also responsible for the life cycle management of NIDS services, including enabling, disabling, and deleting an NIDS service, using a similar procedure.

If a command of type ② arrives at the plugin, the NIDSaaS plugin will notify the plugin agent on the network node of NIDS rule set changes. The agent will update the rule sets stored locally and restart the NIDS processes to apply those changes. When malicious traffic triggers the NIDS rules, an alert file will be generated and stored in the tenant directory on the network node. The size of the alert file depends on the number of rules being triggered.

Finally, if the NIDSaaS plugin receives a command of type ③, the RPC messages that contain the NIDS service ID and NIDSaaS flow information will be sent to the NIDSaaS plugin agents on compute nodes. After receiving the messages, the NIDSaaS OVS agent extension will invoke the OVS driver to set up the bridge and OpenFlow rules, which can mirror the network traffic of the tenant network and then direct the mirrored traffic to the NIDS Neutron port on the network node.

B. Network Model

Fig. 3 illustrates NIDSaaS's network model. We utilize the TaaS as the port mirroring tool. An OVS Tap Bridge *br-tap* is created in each node and connected with the OVS Integration Bridge *br-int* and the OVS Tunnel Bridge *br-tun*, which can be used for separating from the original network traffic and steering the duplicated network traffic.

When an NIDS service is going to be deployed by NIDSaaS, an NIDS port is created on the *br-int* inside the network node, while the TaaS updates OpenFlow rules for *br-int*, *br-tap*, and *br-tun* respectively. During the initialization of NIDSaaS flows, NIDSaaS plugin agents will configure the OpenFlow rules for *br-int* and *br-tun* on compute nodes via TaaS API.

After the configuration of NIDS service and flow is finished, the traffic in and/or out of the VM instances will be duplicated and tagged with TaaS VLAN tag in the *br-int*. Afterwards, these tagged packets are transferred from the *br-int* to the *br-tap*, and then forwarded to the *br-tun*. The *br-tun* will wrap the packets with VXLAN tag and then forward them to the network node via tunnel interface. When the *br-tun* inside the network node receives the mirrored traffic from the VXLAN

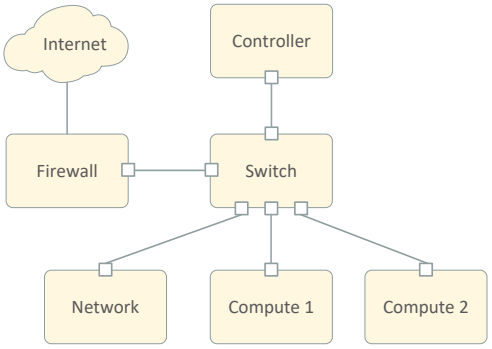


Fig. 4. Testbed Setup

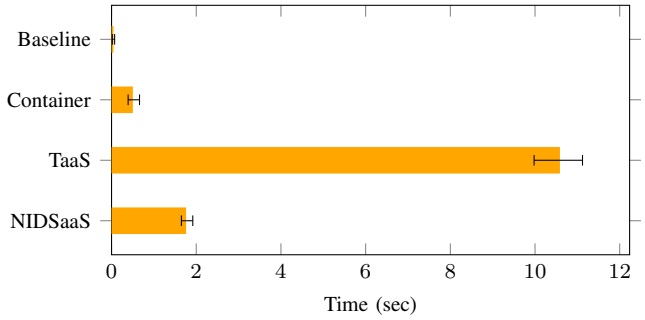


Fig. 5. NIDS Service Launch Time

tunnel, it will unwrap those packets and add the TaaS VLAN tag. After that, the packets will be sent to the *br-tap* and then to the *br-int*. The *br-int* will replace the TaaS VLAN tag with NIDS port tag and deliver the packets to the NIDS port.

IV. Evaluation

In this section, we evaluate the performance of NIDSaaS in terms of the service launch time, and CPU/memory usage under a specific network traffic load in an OpenStack cloud.

A. Experiment Environment

We used four bare-metal nodes in the Chameleon cloud [23] for evaluation. Each of the nodes is equipped with two 2.60 GHz Intel Xeon (24 cores) and 192 GiB RAM. One node works as an OpenStack controller that supplies API, scheduling, and other shared services to the cloud. One node serves as the network node, providing networking service to virtual instances and devices. The remaining two nodes are compute nodes that are installed with OpenStack Nova and Neutron OVS agents. All those bare-metal nodes in the Chameleon cloud are connected to a 10 Gbps VLAN. The topology of the test environment is shown in Fig. 4. The OpenStack services are deployed using the DevStack, which is a series of scripts for quickly launching an OpenStack platform. We chose Snort 2.9.11, configured with 3881 community rules, as the NIDS service for all the test scenarios.

To simulate network attacks and test the functionality of the NIDSaaS, we use a network trace from the Mid-Atlantic Collegiate Cyber Defense Competition (MACCDC) [24]. During each test, we launch a traffic sender VM and a traffic receiver VM. The network traffic is sent from the sender VM to the receiver VM using *tcpreplay*[1]. For our experiments, we modified the trace, e.g., removing IPv6 packets and all VLAN tags and modifying destination MAC addresses. The trace after modification contains 0.99 GiB of data, 3,581,081 packets and it lasts 53 minutes. We also launch a separate VM for NIDS in the TaaS test cases. Each VM launched in our experiments has a 4-core virtual CPU and 4 GiB RAM. The CPU and memory usage is collected by *pidstat*[2] continuously during the tests. In the container tests, we run a Docker container with an NIDS image and record the container resource usage by the *docker stats* command.

We compare the launch time, CPU and memory usage in the following scenarios.

- **Baseline**: Launch the NIDS service in an OpenStack compute node and sniff network traffic from the target network directly.
- **Container (Docker)**: Launch a container-based NIDS service in an OpenStack compute node and monitor target network interface directly, which means the service can only monitor the VMs deployed inside the compute node.
- **TaaS**: Launch an NIDS-enabled VM in the OpenStack. The target tenant network traffic is mirrored to this VM using TaaS so that NIDS can sniff the network traffic from different hypervisors.
- **NIDSaaS**: Deploy and launch the NIDS service on the OpenStack network node directly. The NIDS service will sniff on a designated Neutron port to which the target network traffic is mirrored.

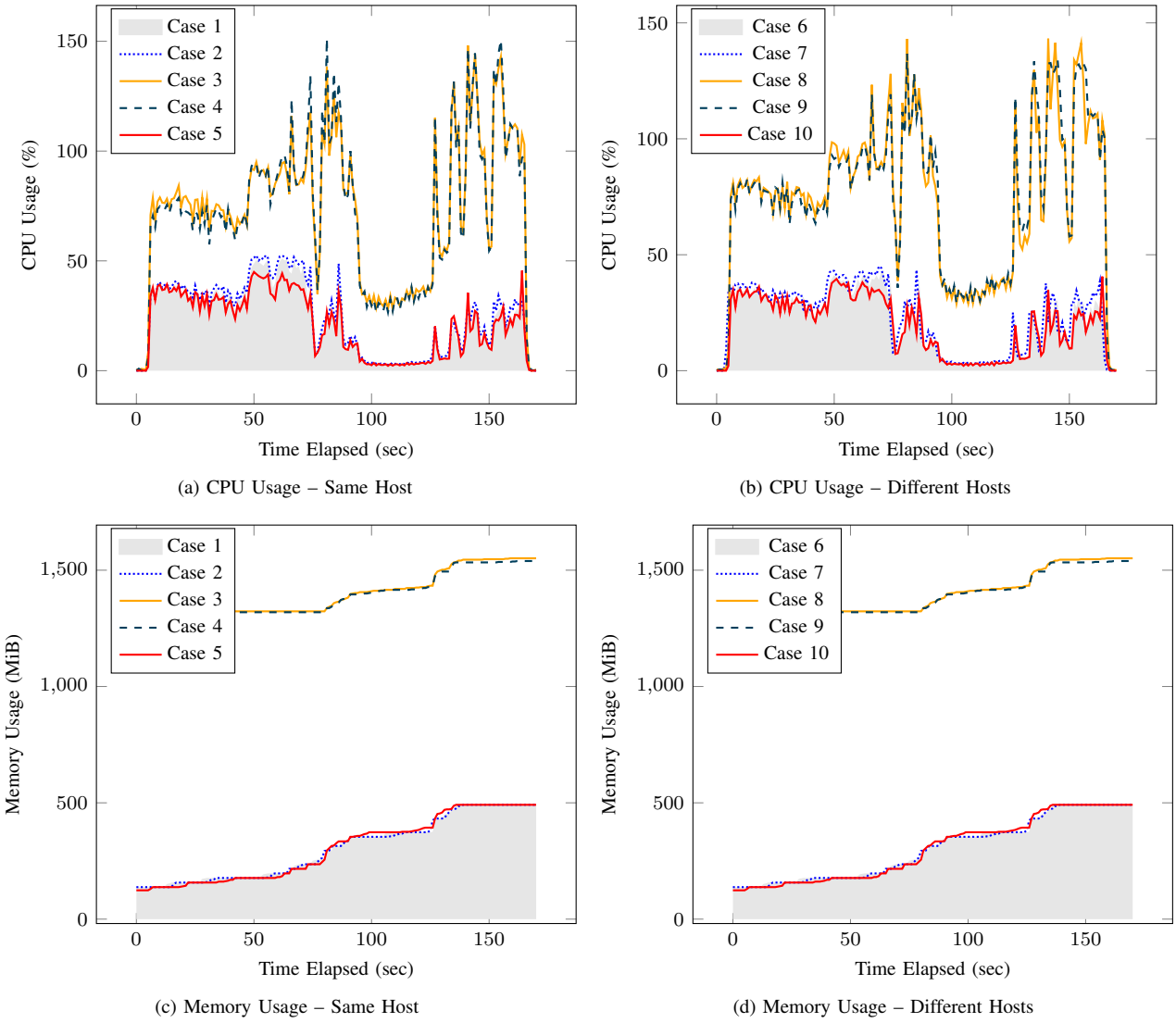
B. Launch Time

To handle the ever-changing volume of network traffic, the NIDS for the cloud should scale quickly. In this experiment, we compare the launch time of NIDS services provisioned by NIDSaaS with the launch time in baseline, container and TaaS scenarios. The launch time is defined as the minimum time required from starting an NIDS service to the point where the service is ready for intrusion detection. In the baseline, the launch time is the duration for an NIDS process to be started and ready. In the container scenario, the launch time refers to the time interval from *docker run* command being executed to the point where the NIDS process inside the container is launched. In the TaaS scenario, the launch time refers to the time needed for booting the NIDS VM and launch the NIDS process. In the NIDSaaS scenario, the launch time is measured from when a user request is sent to the point when the NIDS service is fully started.

Each test is repeated ten times, and the results of their averages are shown in Fig. 5. The average launch time for

[1]https://github.com/appneta/tcpreplay

[2]https://linux.die.net/man/1/pidstat



(a) CPU Usage – Same Host

(b) CPU Usage – Different Hosts

(c) Memory Usage – Same Host

(d) Memory Usage – Different Hosts

Fig. 6. CPU and Memory Usage

the baseline scenario is minimal, only 45 ms, due to no cost in virtualization or communication. The average launch time for the container scenario is 502 ms, ten times larger than the baseline. Since the request of creating an NIDS service needs to be transferred and processed across multiple nodes in the NIDSaaS scenario, the launch time is relatively long, 1.760 seconds on average. However, compared to the TaaS scenario, in which the average is 10.588 seconds, NIDSaaS takes much shorter time (only 16%) to launch, which makes NIDSaaS more "elastic". Launching an NIDS VM is needed in the TaaS scenario, and booting a VM takes much longer time than processing and network communication in the NIDSaaS scenario or starting a container in the container scenario.

C. CPU and Memory Usage

We break down each scenario into multiple cases based on whether the sender and receiver VMs are in the same hypervisor. By doing so, we can evaluate whether the placement of the receiver has an impact on NIDS performance. Moreover, the placement of the NIDS VM in the TaaS scenario also needs

TABLE I
TEST CASES FOR DIFFERENT VM AND NIDS PLACEMENTS

#	Name	Sender	Receiver	NIDS
1	Baseline	Compute 1	Compute 1	Compute 1
2	Container	Compute 1	Compute 1	Compute 1
3	TaaS	Compute 1	Compute 1	Compute 2
4	TaaS	Compute 1	Compute 1	Compute 1
5	NIDSaaS	Compute 1	Compute 1	Network
6	Baseline	Compute 1	Compute 2	Compute 1
7	Container	Compute 1	Compute 2	Compute 1
8	TaaS	Compute 1	Compute 2	Compute 2
9	TaaS	Compute 1	Compute 2	Compute 1
10	NIDSaaS	Compute 1	Compute 2	Network

to be considered. All the placement cases are listed in Table I.

For each test case, we replay MACCDC trace from the sender to the receiver. Given that the actual sending rate can vary for high sending rates, the sending rate is fixed at 50 Mbps in all tests. In each test, the NIDS service issued the same number of alerts (18,388). Packet loss existed but was negligible in those tests. For the baseline, container and NIDSaaS scenarios, we record the CPU and memory usage of the NIDS process once every second during each test. For the TaaS scenario, the CPU and memory consumption of the processes for the NIDS VM in the hypervisor is recorded every second in each test.

The average CPU and memory consumption in all test cases is shown in Fig. 6. Clearly, the baseline cases (1 and 6) consume the least amount of CPU and memory. Since the docker container is lightweight, the container cases are about the same as the baseline and NIDSaaS cases in terms of memory usage and consume around 4% more CPU than the baseline and NIDSaaS cases. In the TaaS test cases, it is evident that CPU usage sometimes exceeds 100%. This is attributed to that more than one CPU core are being used by the NIDS VM. Compared to TaaS, NIDSaaS consumes way less resources, approximately only 7-50% of the CPU time and 9-30% of the memory consumed by TaaS, no matter whether the sender and receiver VMs are on the same host or not. Overall, NIDSaaS shows similar CPU and memory usage patterns as baseline and introduces far less CPU and memory overheads than TaaS.

V. Conclusion

In this paper, we presented a new NIDS service model named NIDSaaS for OpenStack clouds, which provides efficient intrusion detection services to cloud tenants. NIDSaaS leverages the existing OpenStack network infrastructure model and OpenFlow to capture network traffic of one or multiple monitoring targets in an easy and flexible manner. NIDSaaS can instantiate, modify, and terminate NIDS services conveniently. We implemented an NIDSaaS prototype and evaluated its performance using a multi-node OpenStack testbed. Compared with the VM-based NIDS service approach, our approach has way less resource overhead and significantly shorter service provision time. In future work, we will investigate the scalability of NIDSaaS and expand its NIDS support beyond Snort. We will also consider optimizing NIDS service placement and automatic service scaling.

Acknowledgement

The development and evaluation of NIDSaaS used the NSF funded Chameleon cloud.

References

[1] "2019 Data Breach Investigations Report," 2019. [Online]. Available: https://enterprise.verizon.com/resources/executivebriefs/2019-dbir-executive-brief.pdf

[2] P. Mishra, E. S. Pilli, V. Varadharajan, and U. Tupakula, "Intrusion detection techniques in cloud environment: A survey," *Journal of Network and Computer Applications*, vol. 77, pp. 18–47, 2017.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. P. andLarry Peterson, J. Rexford, S. Shenker, and J. Turner., "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, 2008.

[4] "Using the Cisco Span Port for San Analysis," 2015. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/san-consolidation-solution/net_implementation_white_paper0900aecd802cbe92.html

[5] "Open vSwitch Doc: Basic Configuration," 2019. [Online]. Available: http://docs.openvswitch.org/en/latest/faq/configuration/#basic-configuration

[6] "OpenStack Docs: TAP as a Service (TAPaaS)," 2018. [Online]. Available: https://docs.openstack.org/dragonflow/latest/specs/tap_as_a_service.html

[7] K. Scarfone and P. Mell, "Guide to Intrusion Detection and Prevention Systems (IDPS)," *NIST Special Publication 800-94*, 2007.

[8] "Snort Network Intrusion Detection & Prevention System." 2019. [Online]. Available: https://snort.org/

[9] "Suricata IDS." 2018. [Online]. Available: https://suricata-ids.org/

[10] P. Garca-Teodoro, J. Daz-Verdejo, G. Maci-Fernndez, and E. Vzquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Journal of Computers and Security*, vol. 28, pp. 18–28, 2009.

[11] "The Zeek Network Security Monitor." 2019. [Online]. Available: https://www.zeek.org/

[12] H. A. Kholidy, A. Erradi, S. Abdelwahed, and F. Baiardi, "HA-CIDS: A Hierarchical and Autonomous IDS for Cloud Systems," in *2013 Fifth International Conference on Computational Intelligence, Communication Systems and Networks*. IEEE, 2013, pp. 179–184.

[13] Y. Meng, W. Li, and L.-F. Kwok, "Design of Cloud-Based Parallel Exclusive Signature Matching Model in Intrusion Detection," in *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*. IEEE, 2013, pp. 175–182.

[14] C.-H. Lin, C.-W. Tien, and H.-K. Pao, "Efficient and effective NIDS for cloud virtualization environment," in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*. IEEE, 2012, pp. 249–254.

[15] H. Li, H. Hu, G. Gu, G.-J. Ahn, and F. Zhang, "vNIDS: Towards Elastic Security with Safe and Efficient Virtualization of Network Intrusion Detection Systems," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 17–34.

[16] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection," in *Network and Distributed System Security Symposium 2018*, 2018.

[17] L. De Carli, R. Sommer, and S. Jha, "Beyond pattern matching: A concurrency model for stateful deep packet inspection," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 1378–1390.

[18] G. Kim, S. Lee, and S. Kim, "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection," *Expert Systems with Applications*, vol. 41, no. 4, pp. 1690–1700, 2014.

[19] E. K.Viegas, A. O.Santin, and L. S.Oliveira., "Toward a reliable anomaly-based intrusion detection in real-world environments," *Computer Network*, vol. 127, pp. 200–216, 2017.

[20] C. Yin, Y. Zhu, J. Fei, and X. He., "A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks," *IEEE Access*, vol. 5, pp. 21 954 – 21 961, 2017.

[21] Y. Fu, F. Lou, F. Meng, Z. Tian, H. Zhang, and F. Jiang., "An Intelligent Network Attack Detection Method Based on RNN," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*. IEEE, 2018, pp. 483–489.

[22] C. Xu, J. Shen, X. Du, and F. Zhang., "An Intrusion Detection System Using a Deep Neural Network With Gated Recurrent Units," *IEEE Access*, vol. 6, pp. 48 697 – 48 707, 2018.

[23] K. Keahey, P. Riteau, D. Stanzione, T. Cockerill, J. Manbretti, P. Rad, and R. Paul, "Chameleon: A Scalable Production Testbed for Computer Science Research," *Contemporary High Performance Computing*, vol. 3, 2017.

[24] "MACCDC Dataset," 2012. [Online]. Available: https://download.netresec.com/pcap/maccdc-2012/maccdc2012_00008.pcap.gz