

SDN-assisted containerized security and monitoring components

Miguel Borges de Freitas, Pedro Quitério, Luis Rosa, Tiago Cruz and Paulo Simões

Department of Informatics Engineering

University of Coimbra

Email: {miguelbf, pmlour, lmrosa, tjacruz, psimoes}@dei.uc.pt

Abstract—Quite often, the deployment of components for network monitoring or security purposes constitutes a burden, due to the need for IT teams to perform on-site setup procedures, and/or to manually configure diversified equipment or services. While this approach may still be somehow manageable for contained infrastructures such as LANs, it cannot cope with the scale of certain telecommunications service provider or industrial IoT environments.

This paper proposes a solution for flexible virtualized component deployment, which takes advantage of containerized probes, together with SDN-assisted network traffic steering mechanisms. In this scope, the integration of SDN and Docker containers constitutes a relevant development, enabling the creation of probe deployment mechanisms which dismiss the need for physical appliance configuration and/or deployment.

Index Terms—SDN, NFV, Network Security, Network Monitoring, Docker

I. INTRODUCTION

The enforcement of security policies through the deployment of security and monitoring components (such as network intrusion detection systems, honeypots, among others) in traditional network architectures, is often a complex and error-prone task. The network ecosystem is extremely complex, composed of several multi-vendor physical devices which are statically deployed in the network and configured on a per-device level [1]. As the number of network hosts scales (which is usually the case for highly distributed Internet of Things environments) the physical placement of hardware-based network probes becomes unpractical – requiring more flexible and easy-to-manage deployment approaches.

From this perspective, technologies such as Software Defined Networking (SDN) and Network Function Virtualization (NFV) can be regarded as potential enablers for alternative security component deployment models [2]. SDN (using the OpenFlow protocol [3]) makes it possible to implement a flexible data plane management approach, with network programmability and control abstractions being managed from a logically centralized but physically distributed control entity (the network controller). By its turn, NFV shifts network packet processing from dedicated hardware to virtualized computational instances consolidated in commercial-off-the-shelf (COTS) hypervisors.

However, hosting the NFV Virtual Network Function instances poses a challenge by itself. Conventional Virtual

Machines hosted in type-1 hypervisors have a significant overhead, which hampers the scalability of NFV-based approaches. In this scope, container technology, particularly Docker containers are increasingly being regarded by many as an alternative solution due to faster startup speed, lower cost, and improved flexibility compared with hypervisor-based virtualization [4]. Such characteristics make it suitable for the deployment of security and monitoring components in the scope of an SDN/NFV compliant architecture. However, the integration between Docker-based containerization and distributed SDN architectures is yet to be adequately addressed. Most of the existing solutions either promote the integration at a higher level (e.g., using OpenStack [5]) or create external software components that manage both the controller and the Docker engine itself [6].

In this paper we explore an alternative approach by creating a group of SDN applications that take advantage of the low-latency northbound interfaces of the controller and the external interfaces provided by the docker engine – thus keeping the network controller as the primer element of the architecture.

We virtualized two types of the most common traditional network security components: vNIDS (containerized Network Intrusion Detection System) and vHoneyPot (virtual honeypot) on top of the developed architecture. These components are completely independent from the image that is running on the container. For evaluation purposes we virtualized Snort [7] and Tpdump [8] as examples of the vNIDS service and Conpot [9] for the vHoneyPot service. Container images are stored on a secure docker registry (running the same virtualization infrastructure) and accessible to the SDN controller as a catalog of security image services. We also explain how the integration of a distributed controller (ONOS [10]) and the docker engine can be accomplished.

II. RELATED WORK

The use of security detection and prevention mechanisms such as Network Intrusion Detection Systems (NIDS), Honeypots and scan filters, connected to an Openflow-capable switch (via a mirror port), was proposed by [11], using the FloodLight [12] SDN controller. In a similar way, [13] also resorts to a mirroring port, using a load balancer to distribute packets between multiple IDS to avoid SDN controller overflow.

Regarding Honeypots, [14] leveraged the SDN network programming capabilities to dynamically select the most desirable

connection among multiple honeypots. Also, [15] describes a prototype for a honeynet based on an SDN architecture to classify and redirect different classes of attacks, such as denial of service and network spoofing, by analysing each packet header at the controller level.

Cziva et al. [16] propose a framework comprised of several container-based (Docker) Network Functions, based on third-party open-source tools such as NIDS (Snort), a packet filter (Scapy), a firewall (iptables), a traffic control (Tc) and a load balancer (Scapy) connected to OpenDaylight SDN controller. The lightweight container-based approach allowed them not only to improve the deployment latency but also provides several other advantages inherent to this technology. The solution though lacks clustering at the virtualization pool level and delegates all the orchestration to an external agent, not taking into consideration the low latency interfaces provided by the controller at the application layer.

Also regarding container-enabled approaches, the OpenvSwitch project [17] provides a small bash script which is able to attach an existing Docker container to an OpenvSwitch bridge. [18] proposes an architecture based on the Ryu [19] SDN controller, which was designed to redirect network traffic to an Intrusion Detection and Prevention System (Snort) instances hosted on Docker containers (the deployments were performed using the Docker daemon). Finally, [20] describes an approach based on the ONOS controller to recreate a hierarchical and collaborative NIDS architecture based on Docker containers and Containernet [21].

Despite several literature proposals that leverage SDN and NFV together with a container-based approach, most of them are merely conceptual. Hence, there is a gap regarding documented practical implementations that can fully explore the clustering characteristics of a modern and production-ready SDN controller architecture, together with the integration of a container-based virtualization pool.

III. PROPOSED ARCHITECTURE

Fig. 1 presents the generic architecture for the proof-of-concept prototype that was implemented. We took advantage of the convenient parallelization mechanisms of the ONOS distributed SDN controller to form a 3 node cluster able to program and manage an SDN network formed by three hosts. The data plane of the SDN network was the responsibility of an OpenvSwitch (OVS) server running on COTS hardware with 6 physical interfaces. Five of those interfaces were configured to use the Data Plane Development Kit (DPDK) [22] userspace driver thus bypassing any packet processing and copy operations that would otherwise occur within the Linux Kernel - promoting the use of DMA transfers to enable direct access to packets from the NIC buffer, maximizing the software switch packet throughput.

Two virtualization nodes are connected to the OVS switch (see Fig. 3) for virtual probe deployment, each one using 2 physical NICs - 1 directly connected to the OVS (for containers to access the SDN network) and another connected to the management VLAN (for container orchestration). The

OVS switch has the mastership assigned to one controller node while maintaining redundant connections to the other nodes for increased availability and load balancing. A management web interface orchestrates the SDN controller cluster and any configuration options that vProbes expose.

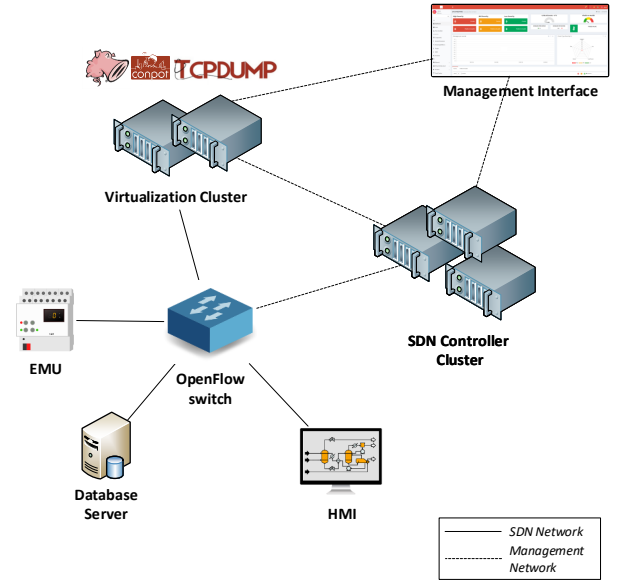


Fig. 1. SDN / NFV Reference Scenario.

The network controller hosts four custom-developed JAVA/OSGi-based applications, together with other other two applications provided by the controller core itself (see Fig. 2):

- **vNIDS** - An application responsible for pro-actively installing flow rules on the switch fabric (OVS in this case) to achieve traffic steering. The application obtains the network graph from the ONOS `TopologyService` and calculates the shortest path between the host to be monitored and the launched container probe host. For each switch in the path, it installs a rule (`FlowObjectiveService`) to output any traffic having the source or destination mac address of the host to the respective port leading to the container.
- **vHoneypot** - An application that takes advantage of ONOS `IntentService` to create host-to-host intents with the container probe and all the other physical hosts on the network - thus making the container a standard host in the SDN network.
- **Docker** - An application that consumes the Docker REST interface exposed from the virtualization hosts via a Transport Layer Security (TLS) socket. It is responsible for registering virtualization nodes in the ONOS distributed database (MapDB), starting containers, uploading images to the private docker registry and identifying launched containers as standard ONOS `Host` objects.
- **WebAPI** - Exposes the aforementioned OSGi application services via REST (stateless operations) or Websocket (real time data) endpoints, used by the management UI.

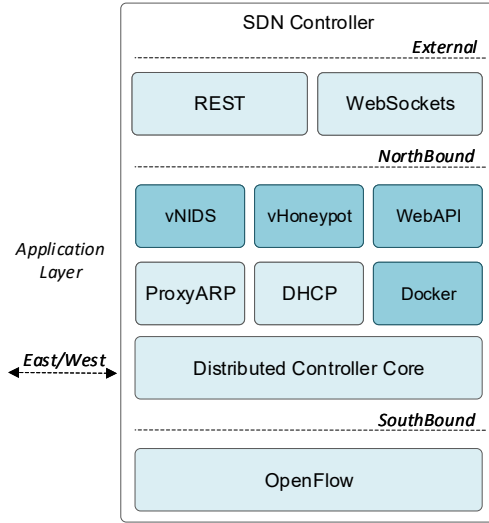


Fig. 2. SDN Controller Architecture.

The ProxyARP application is important to the architecture so that the controller is able to detect containers being launched due to their inherent ARP packet broadcasts. The DHCP application effectively virtualizes the DHCP service - assigning IP addresses to containers (vHoneyPot) on the same subnet of the physical SDN hosts. Although the controller supports a broad range of protocol providers at the controller southbound interface, only OpenFlow was enabled.

It is important to understand how containerization (Docker) can be integrated in an OpenFlow-based SDN network. As of today, the Docker engine lacks any support for the OpenFlow protocol [23]. To solve this issue, we installed OpenvSwitch at each virtualization node, thus providing an OpenFlow bridge in addition to the standard network bridge already provided by the docker default network mode (see Fig. 3). We then attached the virtualization node physical NIC that is connected to the OVS switch (see Fig. 1) to this new bridge - resulting in a path to the SDN network. The OpenFlow bridge was configured similarly to the OVS switch - with its mastership assigned to one of the nodes of the controller cluster (with fallbacks to other nodes). Additionally we developed an external agent (SDN glue agent) configured to listen to any events emitted by the docker engine. When a new container is launched via the Docker controller application, the agent introspects the resulting event for the presence of a specific label (`--sdn=true`) and gets the container id.

Upon the existence of the aforementioned label and considering that container isolation is achieved via cgroups [24] and network namespaces [25], the agent then automatically creates a linux virtual interface (*vif*) and effectively establishes a virtual ethernet interface pair (*veth*) by:

- 1) placing the *vif* in the container network namespace (making it available to the container) and
- 2) attaching the *vif* to the OpenFlow bridge (making it available in the SDN network).

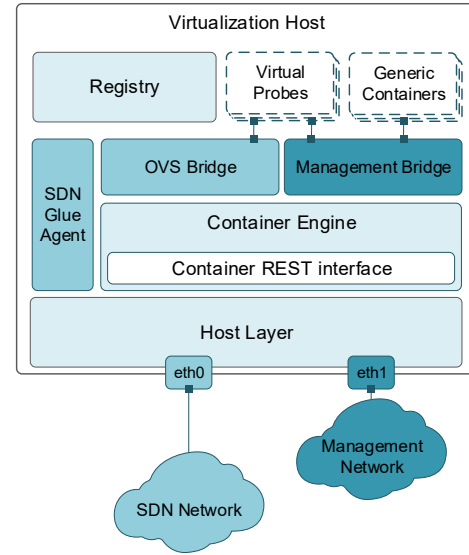


Fig. 3. Virtualization Host Architecture.

With this approach, the developed prototype was able to seamlessly integrate containers in the SDN network while providing the means for the co-existence of generic containers/services (not attached to the OpenFlow bridge) in the same virtualization node - contributing to computational resource optimization within the virtualization pool. In fact, the registry (where virtual probe container images are stored) is an example of a container that is accessible to the management network but not to the SDN network.

While container orchestration frameworks (such as kubernetes) could be leveraged to manage container pools, these often manage their own networking. SDN controller integration could be possible, but it is a complex matter that may affect the neutrality of the proposed solution, also introducing potential performance issues.

IV. EXPERIMENTAL EVALUATION

An integrated scenario based on aforementioned architecture (see Fig. 1, 2 and 3) was devised to represent a small industrial automation environment protected by a set of SDN assisted detection probes, deployed as Docker containers.

The complete scenario includes: an SDN controller cluster (from 3 to 5 nodes) hosted on a VMware ESXi 6.5 hypervisor, with 8 vCPUs (Intel Xeon CPU @ 2.80GHz) and 30GB RAM per node; an OpenFlow switch running CentOS 7, configured with Intel DPDK support and hosted within a Dell Poweredge R210 server with 8 GB of RAM; two virtualisation servers (with Intel Xeon E3110 dual-core CPUs and 8GB RAM each) configured with the Docker engine and OpenvSwitch.

Three additional hosts were also configured, namely: a custom-developed Environmental Monitoring Unit (EMU), designed to collect humidity and room temperature measurements from our laboratory datacentre, as well as two ESXi VMs - a database server and a Human Machine Interface

(HMI), the latter being supported using the RapidSCADA software (which polls the EMU registers to get telemetry updates), both attached to the OpenvSwitch using native network interfaces (ESXi passthrough mode).

The first test consisted in deploying each probe (vNIDS and vHoneybot) to the virtualization server 25 times while measuring how much time the internal setup operations account for within the overall deployment latency (see Fig. 4). For the vNIDS, there are no DHCP SDN assignments as the vProbe does not have any address in the SDN network. Whereas, for the vHoneybot, the container needs to acquire a valid IP address. No significant differences were observed, both of them were launched in around 4.5 seconds. Moreover, the bulk of this time was devoted to container startup rather than SDN network programming or additional DHCP setup.

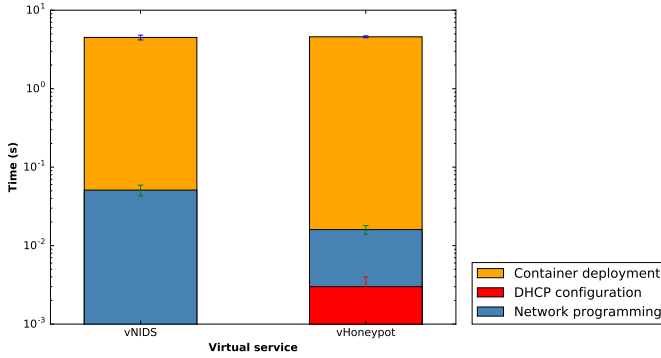


Fig. 4. Time taken for each SDN probe deployment step (3 host network).

The second test consisted in measuring the time taken to program the network to add a variable number of hosts to the monitoring list of a previously deployed vNIDS. Additionally, for each trial it was used a variable number of controller nodes (from 1 to 5). The results, illustrated in Fig. 5, show that the cluster size does not severely impact the network programming times. For the largest scenario, a 5 node controller cluster, we were able to add 100 hosts to the vNIDS in around 4 seconds.

The third experiment, consisted of measuring the raw data plane bandwidth between hosts using the *iperf2* tool, both in TCP and UDP modes, and compare those values against the bandwidth obtained from the copy of the network traffic measured at the vNIDS *vif* using the *iptraf* tool. The results demonstrated in Fig. 6 show a performance comparable with commercial switches with the same specifications (1 Gbps), despite the use of an SDN-based approach. The values measured at the vProbe are similar to the raw TCP measurements - so we can conclude that the proposed solution does not negatively impact the NIDS packet throughput performance.

V. CONCLUSIONS

The presented architecture leverages the advantages of SDN/NFV, together with Docker-based component containerization to provide security services for protecting a distributed infrastructure, constituting a highly flexible approach for network security management and monitoring.

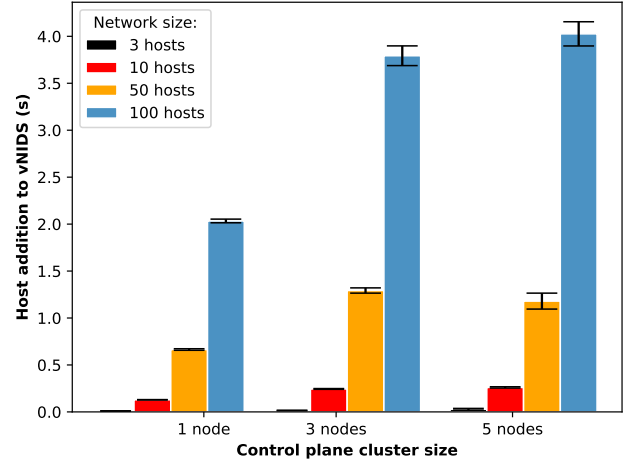


Fig. 5. Host addition to a deployed vNIDS service depending on the overall network and control plane cluster sizes.

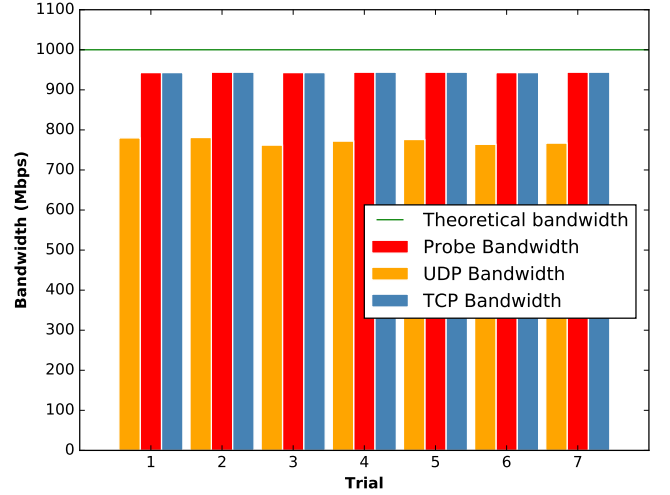


Fig. 6. Raw TCP / UDP performance vs vNIDS bandwidth.

Experimental evaluation revealed the SDN data plane and the traffic-steering flows to exhibit packet throughput performance which is comparable with traditional counterparts. Moreover, virtual probe deployment is achieved in approximately 4 secs, with the Docker container startup accounting for the majority of the overall time-frame, which represents a clear improvement over the deployment latencies characteristic of traditional virtual machine-based implementations. The Docker registry can be used as part of the architecture to create a catalog of available container probes while taking advantage of the already provided built-in Docker security update mechanisms (e.g., the update framework).

ACKNOWLEDGMENT

This work was partially funded by the Mobiwise (POCI-01-0145-FEDER-016426) and Mobilizador 5G (POCI-01-0247-FEDER-024539) P2020 projects.

REFERENCES

- [1] M. R. B. de Freitas, "Network softwareization for IACS security applications," Master's thesis, Universidade de Coimbra, 2018. [Online]. Available: <https://estudogeral.sib.uc.pt/handle/10316/83548>
- [2] M. B. de Freitas, L. Rosa, T. Cruz, and P. Simões, "SDN-Enabled virtual data diode," in *Computer Security*, S. K. Katsikas, F. Cuppens, N. Cuppens, C. Lambrinouidakis, A. Antón, S. Gritzalis, J. Mylopoulos, and C. Kalloniatis, Eds. Cham: Springer International Publishing, 2019, pp. 102–118.
- [3] "OpenFlow Switch Specification," 2015. [Online]. Available: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>
- [4] Z. Wang, D. Wu, Z. Luo, and Y. Du, *Building a Lightweight Container-Based Experimental Platform for HPC Education*. Springer, 01 2020, pp. 175–183.
- [5] A. Celesti, A. Levin, P. Massonet, L. Schour, and M. Villari, "Federated networking services in multiple openstack clouds," in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2015, pp. 338–352.
- [6] F. Moradi, C. Flinta, A. Johnsson, and C. Meirosu, "ConMon: An automated container based network performance monitoring system," *Proceedings of the IM 2017 - 2017 IFIP/IEEE International Symposium on Integrated Network and Service Management*, pp. 54–62, 2017.
- [7] Cisco, "Snort - network intrusion detection and prevention system." [Online]. Available: <https://www.snort.org/>
- [8] The Tcpdump Group, "Tcpdump/libpcap public repository." [Online]. Available: <https://www.tcpdump.org/>
- [9] Conpot development team, "Conpot ics/scada honeypot." [Online]. Available: <http://conpot.org/>
- [10] "ONOS - A new carrier-grade SDN network operating system designed for high availability, performance, scale-out." 2020. [Online]. Available: <https://onosproject.org/>
- [11] N. F. Huang, C. Wang, I. J. Liao, C. W. Lin, and C. N. Kao, "An OpenFlow-based collaborative intrusion prevention system for cloud networking," in *Proceedings of 2015 IEEE International Conference on Communication Software and Networks, ICCSN 2015*, 2015, pp. 85–92.
- [12] "Open Source Software for Building Software-Defined Networks," 2020. [Online]. Available: <http://www.projectfloodlight.org/floodlight/>
- [13] Monshizadeh *et al.*, "Detection as a service: An SDN application," *International Conference on Advanced Communication Technology, ICACT*, pp. 285–290, 2017.
- [14] W. Han, Z. Zhao, A. Doupe, and G. Ahn, "HoneyMix: Toward SDN-based Intelligent Honeynet," *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pp. 1–6, 2016. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2876022>
- [15] A. Manzano, J. Naranjo, I. Bernal, and D. Mejía, "A prototype for a honeynet based on SDN," in *2016 8th Euro American Conference on Telematics and Information Systems, EATIS 2016*, 2016.
- [16] R. Cziva, S. Jouet, K. J. White, and D. P. Pezaros, "Container-based network function virtualization for software-defined networks," in *Proceedings - IEEE Symposium on Computers and Communications*, vol. 2016-February, 2016, pp. 415–420.
- [17] "OVS-Docker Github," 2017, accessed: 2017-12-7. [Online]. Available: <https://github.com/openvswitch/ovs/blob/master/utilities/ovs-docker>
- [18] T. G. Nguyen, T. V. Phan, B. T. Nguyen, C. So-In, Z. A. Baig, and S. Sanguanpong, "Search: A collaborative and intelligent nids architecture for sdn-based cloud iot networks," *IEEE Access*, vol. 7, pp. 107 678–107 694, 2019.
- [19] "Ryu component-based software defined networking framework," 2020. [Online]. Available: <https://github.com/osrg/ryu>
- [20] J. Xing, H. Zhou, J. Shen, K. Zhu, Y. Wangt, C. Wu, and W. Ruan, "Asidps: Auto-scaling intrusion detection and prevention system for cloud," in *2018 25th International Conference on Telecommunications (ICT)*, June 2018, pp. 207–212.
- [21] M. Peuster, H. Karl, and S. van Rossem, "Medicine: Rapid prototyping of production-ready network services in multi-pop environments," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016, pp. 148–153.
- [22] Linux Foundation, "DPDK project." [Online]. Available: <https://www.dpdk.org/>
- [23] "Docker Network Overview," 2020. [Online]. Available: <https://docs.docker.com/network/>
- [24] T. Heo, "Control group v2." [Online]. Available: <https://github.com/torvalds/linux/blob/master/Documentation/admin-guide/cgroup-v2.rst>
- [25] T. Combe, A. Martin, and R. Pietro, "To docker or not to docker: A security perspective," *IEEE Cloud Computing*, vol. 3, pp. 54–62, 09 2016.