D-STREAMON - NFV-capable distributed framework for network monitoring

Davide Palmisano* Pier Luigi Ventre* Alberto Caponi† Giuseppe Siracusano*
Stefano Salsano*† Marco Bonola† Giuseppe Bianchi*†
*University of Rome Tor Vergata, Italy
†Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Italy
Email: {name.surname}@uniroma2.it

***Abstract*—Several reasons make NFV an attractive paradigm for IT security: lowers costs, agile operations and better isolation as well as fast security updates, improved incident responses and better level of automation. At the same time, the network threats tend to be increasingly complex and distributed, implying huge traffic scale to be monitored and increasingly strict mitigation delay requirements. Considering the current trend of the networking and the requirements to counteract to the evolution of cyber-threats, it is expected that also network monitoring will move towards NFV based solutions. In this paper, we present Distributed StreaMon (D-StreaMon) an orchestration framework for distributed monitoring on NFV network architectures. D-StreaMon has been designed to face the above described challenges. It relies on the StreaMon platform, a solution for network monitoring originally designed for traditional middleboxes. Changes that allow Streamon to be deployed on NFV network architectures are described. The paper reports a performance evaluation of the realized NFV based solutions and discusses potential benefits in monitoring tenants' VMs for Service Providers.**

***Index Terms*—Network Function Virtualization, Network Monitoring, Threat Detection, Network Programmability**

I. Introduction

The unyielding increasing trend, the evolving complexity and the diversified nature of modern cyber-threats [1] calls for new scalable, accurate and flexible solutions for network traffic monitoring and threats detection. New solutions should support zero-touch configurations, flexibility, agility and real-time traffic analysis capabilities to promptly detect and mitigate cyber-attacks. Indeed, the real challenge is to promptly react to the high mutable needs by deploying customizable traffic analyses functionalities, capable of tracking events and detect different behaviours of attacks. Such objectives can be achieved by efficiently handling of the many heterogeneous features, events, and conditions which characterize an operational failure, a network's application mis-behavior, an anomaly or an incoming attack. The flexibility in the network monitoring can be obtained through the systematic exploitation of stream-based analysis techniques, carefully combined to address *scalability-by-design* and made available to the programmer by means of open APIs. Even more challenging, traffic analyses and mitigation primitives should be ideally brought inside the monitored network and the monitoring probes themselves. This avoids the centralization of the analysis which moves flows of data to a central point resulting inadequate to cope with the huge traffic scale and the strict (ideally real-time) mitigation delay requirements. Solutions for the aforementioned challenges find breeding ground in the current trends towards the softwarization of networks [2] [3] which aims at realizing a Software Defined Infrastructure and are driving the creation of data centers as partially replacement of the legacy network infrastructures. Such trends includes technologies like Network Function Virtualization (NFV) [4], Software Defined Networking (SDN) [5] and Cloud Computing [6]. More specifically NFV, focusing on the decoupling of network functions from physical devices on which they run, becomes the ideal technology to bring to the reality the idea of pervasive network monitoring. We expect that fields like network security could benefit in moving towards NFV paradigm, where security systems or hardware based middleboxes are substituted by virtualized network running in commodity servers

In this paper, we propose a distributed framework for wide-spreading network monitoring, based on NFV principles where the network probes run in commodity servers leveraging on lightweight virtualization technologies. We call this framework D-Streamon, as for the monitoring engine we used StreaMon [7] a data-plane programming abstraction for stream-based monitoring tasks. Modern cyber-threats can easily break the controls performed by the standard procedures which aims at monitor the perimeter of an infrastructure and call for a innovative approaches in the defense techniques and for the deployment of a widespread monitoring. For this reason we develop a distributed framework where the StreaMon probes can communicate between each other and provide information to external frameworks. This can enable advanced monitoring techniques based on the correlation of the information and overcome the problem derived of the single point of control. The costs to realize such pervasive infrastructure, are prohibitive and no more affordable using hardware-based middle-boxes: they require significant capital investment and the procedures to maintain are cumbersome and error prone. Therefore, we completely rethink the lifecycle of a StreaMon probe and we adapt StreaMon to run as "containerized" software based probe, which open to scenarios envisaged by the NFV paradigm where the hardware based middleboxes are replaced by virtualized network appliances running on COTS hardware. To complete our framework,

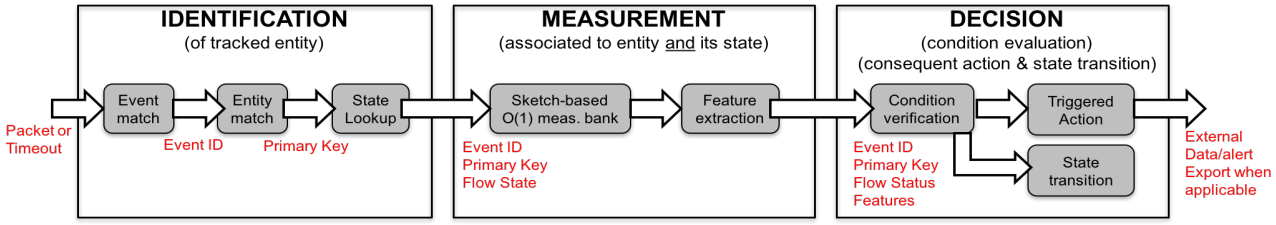


Fig. 1: StreaMon data plane identification/measurement/decision abstractions

we realized a set of management tools with a GUI which makes agile and flexible the deployment of the StreaMon based probes: the repetitive, not scalable and error prone prcodures are replaced by a software based orchestrator.

The contributions of this paper are:

- the adaption of StreaMon solution to the lightweight virtualized environments;
- the realization of two "virtualized" variants, *Processes oriented* and *Containers oriented*;
- the performance evaluation of the NFV based solutions and of the legacy approaches based on the comparison of the resources usage;
- the streamline of the deployment procedure related to StreaMon making possible the integration in software based orchestration;
- the division of the original features of Streamon and the introduction of two separate entities, the controller and the probes;
- the introduction of publish/subscribe architecture which enables the communication between StreaMon probes and makes available the information to other frameworks;
- the implementation of a set of management tools enriched with a compelling GUI which ease the deployment of a large set of probes;

An open source implementation can be downloaded from our repositories [8] and is available for evaluation. The paper is structured as follows: Section II elaborate on StreaMon platform; the D-StreaMon solution is described along with the NFV based solutions in Section III; Section IV provides a performance evaluation of the designed solutions; Section V reports on related work; finally in Section VI we draw some conclusions and highlight the next steps.

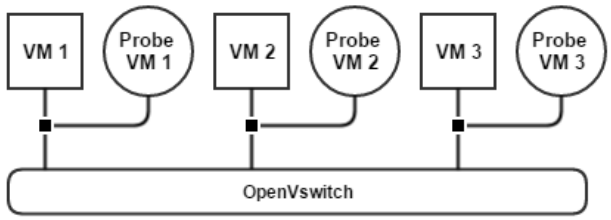
II. StreaMon

StreaMon is a software defined platform for stream-based monitoring tasks directly running inside network probes, which exploits eXtended Finite State Machines (XFSM) for programming custom monitoring logic. StreaMon's strategy closely resembles the one pioneered by Openflow [9] in defining open abstractions of networking functionalities. Such abstractions are programmable by the application's developer leveraging on open API which consists of three "stages". Identification stage permits the programmer to specify what is the monitored entity, i.e. a combination of packet fields, associated to an event triggered by the actual packet under arrival, as well as retrieve an eventually associated state. Measurement stage allows the programmer to configure which information should be accounted. It integrates *efficiently implemented* measurement primitives (metric modules), fed by configurable packet fields, with externally programmed *features*, expressed as arbitrary arithmetic operations on the metric modules' output. Decision stage defines the application logic in the form of eXtended Finite State Machines (XFSM), i.e. check conditions associated to the current state and tested over the currently computed features, and trigger associated actions and/or state transitions.

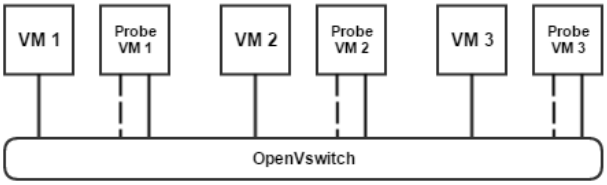
The previously introduced abstraction is concretely implemented by a stream processing engine consisting of four modular layers. Event layer is in charge of parsing each raw captured packet and matches an *event* among those *u*ser-programmed via the StreaMon API. The matched event permits to retrieve an *eventually* stored state. The event layer is further in charge of supplementary technical tasks, such as handling special timeout events, deriving further secondary keys, for further details [7]. The application programmer can instantiate a number of *metrics* derived by a basic common structure (i.e., Bloom-type sketches, d-left hash tables), updated at every packet arrival. This constitutes the Metric layer. Feature layer permits to compute user-defined arithmetic functions over (one or more) metric outputs. Whereas metrics carry out the bulky task of accounting *basic* statistics, the features compute *derived* statistics tailored to the specific application needs. The Decision layer is the final processing stage implements the actual application logic. It keeps a list of *conditions* expressed as mathematical/logical functions of the feature vector provided by the previous layer and any other possible secondary status. Each condition will trigger a set of specified and built-in *actions* and a state *transition*. StreaMon's abstractions are illustrated in Figure 1.

III. D-StreaMon

The original architectural design of StreaMon platform makes it viable to be used in small scale scenarios where a limited number of network monitoring probes is involved. It is not trivial to scale StreaMon to large-scale scenarios in which a high number of probes deployed in the network require to be *orchestrated*. In this Section we elaborate on the evolution of the platform the so called *Distributed*-StreaMon (D-Streamon). It is designed to match modern requirements for the scalability, bringing the StreaMon platform from a standalone implementation to a more NFV-friendly architecture suitable for distributed and heterogeneous environments where



(a) Probes as Processes



(b) Probes as Containers

Fig. 2: D-StreaMon architectures

new-generation threats should be mitigated. The architecture has been initially presented in [10].

A. Challenges

The StreaMon implementation relies on slow deployment process, which is not suitable for large scale scenarios and is not flexible and enough agile to be integrated in software orchestrations. A control and coordination layer able to adaptively orchestrating remote probes is a mandatory requirement. New generation of industrial control systems or critical infrastructures calls for programmable abstractions in the monitoring framework [11]. Another question that should definitely addressed for the next generation monitoring framework is "How to support resources constrained devices and computationally limited VMs ?". StreaMon and in general probes should be able to run on low performance devices like network switches or in light-weight virtual computing resources like Unikernels, Tinified VMs and Containers. This will enable scenarios where the probes are able to run also in small ubiquitous clouds. The legacy StreaMon considered an implementation through high performance middleboxes which monitor a single-point of the network. Network threats tends to be increasingly complex and distributed. The Advanced Persistent Threats (APTs) [12], to give an example, call for large scale deployments, the interconnection and the cooperation of multiple network probes that control different points of the network. On one hand, the interconnection, through chaining, of multiple StreaMon probes for different levels of processing can improve the whole monitoring performances. On the other hand it reduces the resource requirements making viable the deployment in resources limited computing nodes. In this way, probes tend to become small and highly specialized encouraging also the re-usability.

B. Architecture design and implementation

StreaMon has been originally designed to run as specialized middlebox, even though it introduces open abstractions to program the network probe. It provides an high customizable framework enabling the description of the XFSM related to the desired monitoring operations by using an high-level XML-based language. Despite such a flexibility, by design does not consider distributed environment involving an high number of probes, since it foresees for a vertical integrated stack where *Control* and the *Monitoring Probe* run on the same node. The first contribution of this work consists in splitting the functionalities of the original StreaMon platform and distribute them between *Control* and pure *Monitoring* operations. Indeed, inpired by the SDN approach, D-StreaMon introduces a clean separation of concerns giving to a control component the burden of controlling the entire monitoring architecture and leaving to the Probe components the task of executing the network monitoring logic and communicate the feed-backs to the controlling engine.

The D-StreaMon's controller is in charge of managing probes' configurations, described by through the StreaMon XML-based language. These high-level configurations are subsequently used to generate the runnable objects to be pushed and executed in the monitoring probes. In this transition, the Control component is committed to perform also the cross-compilation of the StreaMon runnables. This approach can introduces several benefits in the architecture: i) maintain the monitoring process agnostic to the platform that runs the probe; ii) restrict the operational functions to the controller and limit the number of components that runs on the probe. By using this approach, the monitoring probe machines result to be very light-weight while the controller centralizes many of the functionalities. Even tough the latter seems to be a single point of failure,it can be easily replicated and scale to the required processing capabilities.

The whole management process of the network monitoring probes has been implemented leveraging on the Ansible [13] framework. With no operational overhead and without requiring the presence of agents in the managed hosts (it only relies on a SSH session between the controller and the host), Ansible allows to remotely execute tasks in the host, without the respect of the OS running on the host machine. To support the operator in the management and the supervision of functionalities described above, D-StreaMon provides an intuitive and simple to use management dashboard, based on the open-source alternative to the official Ansible's dashboard Semaphore, that enables to perform the installation, execution and removal of monitoring probes in the host machines.

As regards the communication architecture between probes and the controller, D-StreaMon does not rely on the classical client/server model. Since the monitoring probes can be hierarchically distributed (to process the network flows at different levels) and should even be able to cooperate, D-StreaMon exploits the flexibility of the publish/subscribe networking model. Each monitoring probe acts as an independent publisher following the actions defined in the configuration of the XFSM and publishes events assigning to them a label (namely a *topic*) that permits to identify the type of information forwarded (e.g. info, alert, warning, log, etc.). This allow interested subscribers to filter events based on that topic. Note

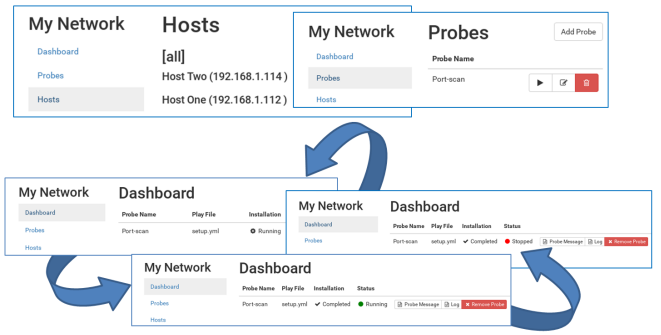


Fig. 3: D-StreaMon deployment tools.

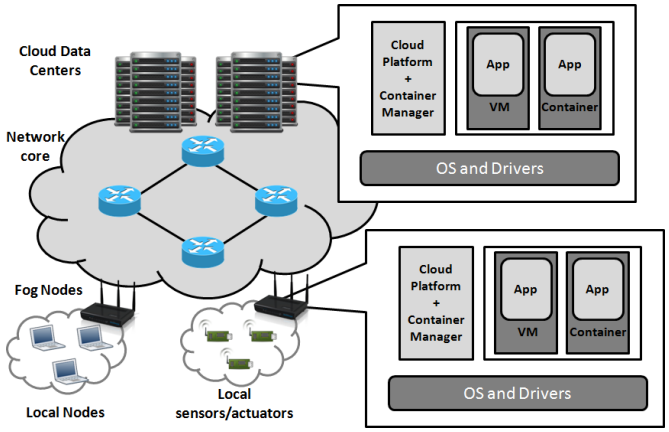


Fig. 4: Target scenarios

that the controller is a *special* subscriber that listens to the whole set of topics created by each monitoring probe. This architecture has been realized by means of the ZeroMQ [14] library. In this Section, it has been described how we re-designed StreaMon and how the deployment procedure has been streamlined making possible the integration in software based orchestration. This was a pre-requisite for the following illustrated in the next Section, where we describe how we adapt StreaMon to the lightweight virtualized environments.

C. Solutions for virtual infrastructures

Nowdays' network trends are driving the creation of data centers as partially replacement of the legacy network infrastructures. In particular, at the ground of these movements there is a general agreement which tends at apply the best practices defined by the Cloud computing to the network infrastructures. In this paper, we focus on StreaMon applicability as NFV framework for infrastructure monitoring in two reference scenarios: Mobile Edge Computing and Cloud Computing. The bottom part of the Figure 4 shows a Fog Computing use case, where the base stations offer connectivity to the mobile clients and have also the computing resources to build small scale data center. The top part represents the classical Cloud Computing scenario, where the providers guarantee IAAS solution to their users. In this scenario, we are interested a solution for the monitoring of the network infrastructure and with security purposes and not monitoring in terms of usage or accounting. In particular, we aim at monitor the activities of the tenants and we propose D-StreaMon which has been adapted to run in virtualized environments. We consider a lightweight virtualization solution based on the so called *Containers*.

Let us consider the migration of the StreaMon platform to a NFV based environment. The StreaMon Probes needs to be instantiated in Cloud data centers or in small scale cloud at the edge of the network with the objective of monitoring the tenants' virtual resources. Using the architecture described in Section III, we have designed two NFV based deployments. The first solution is referred to as *Probes as Processes* (Figure 2a). The monitoring Probes are executed as processes running in the main host process space. The Probes intercept the traffic directly from the ports of the target which have been bridged in the L2 switch of the virtualization server (in our case we use Open vSwitch [15] to implement the Layer 2 switch of the virtualization server). The second solution is referred to as *Probes as Containers* and is represented in Figure 2b. It envisages the use of the Docker Containers [16] for running the probes: for each target to be monitored, a StreaMon Probe is created and deployed in a Docker Container. All Containers are attached to the same L2 switch and the port mirroring functionality is used to "copy" the VM traffic towards the associated Probe. In this solution, the probes need two different network interfaces. One is used to mirror the associated VM traffic (dotted line in Figure 2b); the second interface instead is used to manage the container itself (continuous line). The first NFV solution can provide better performance, as the Process virtualization introduces less overhead with respect to Containers solution and because the packets do not need to be "mirrored" by the switch. At the same time the second solution can provide a better isolation and does not request particular workarounds to have different StreaMon Probes working in several Processes co-located in the same host. Moreover, the Container based solution is more attractive for the availability of different orchestration tools like Docker Swarm [16], Kubernetes [17], Nomad and many others. These solutions can provide valuable improvements to the architecture as they can automate aspects not taken into account by our tools in the deployment of the Probes.

IV. Experimental Results

As preliminary evaluation of the NFV based solutions, we performed a simple benchmark in which the CPU usage is considered for 5 different use cases: i) baseline host; ii) baseline container; iii) probe in the host; iv) probe as process; v) probe as container. The rationale for this evaluation is to provide an indication on the scalability of the approaches for a Virtualization Infrastructure made up of Linux VMs (tenants VMs) and Linux Containers/Processes running on typical Virtualization Servers. Our experimental set-up is composed by two hosts equipped with an Intel i3-2100 dual-core CPU and 8G of RAM. The first host (hereafter named as H_S) is used to generate network traffic and forward it towards the second

host (namely H_C) that receive the packets and make them available to the D-StreaMon monitoring probes running on such host. Both the hosts run on Debian 8.3 operating system with Xen-enabled v3.16.7 Linux kernel and are equipped with two network interfaces at 1 Gb/s, the first interface is used as the management interface and the latter one serves the direct interconnection of the host (data plane network). The CPU load has been obtained collecting the idle percentage through *top* suite. All results have been obtained by executing a test of 100 replicated runs. Error bars in the figures denote the 95% confidence intervals of the results. However, they are so close to the average that are barely visible in Figure 5. During the experiment, the probes receive traffic at three different rate of packet/s respectively 22500 pps, 45000 pps and 68750 pps with a packet size of 100 byte. To generate the traffic we used the *iperf* tool which simultaneously launches four different UDP flows. For each of the three rates, we have evaluated the CPU utilization for the aforementioned cases.

In the *Baseline host* case, there are no monitoring probes active so that a mere evaluation of the introduced overhead is performed. With respect to the previous case, where the *Baseline container* is considered, the *iperf* Server is executed inside a Docker Container. In the *Probe in host* case the StreaMon probe resides in H_S, without including any virtualization mechanism. Obviously, the introduction of the StreaMon probe increases the CPU utilization with a total overhead value near the 35%: high increase is due to the fact that the probe has to analyze the whole received traffic and, by increasing the packets per second rate, the CPU utilization increases more rapidly than the first two columns. This is motivated by the configuration adopted to run StreaMon in this tests, which used *Libpcap* for the processing of the packets. In general, we observe that the increase of the packets rate introduces a linear increment of the CPU load for all the considered use cases. Moving the *iperf* server inside a container (*Baseline container*) introduces about 13.17% of overhead. This is due to the execution of the container itself and the insertion of an Open vSwitch to perform the forwarding of the traffic.

The *Probes as Process* and *Probes as Container* columns show the CPU utilization of the server machine while running a probe using the designed NFV based architectures. As expected, running a probe inside a container is more expensive than running it as a process. The overhead introduced by the container and by the forwarding of the traffic account for about a 13,47% of overhead. Note that the probes in the containers need two interfaces and the forwarding of the traffic is realized through the mirror functionality provided by Open vSwitch. The overhead is stable when varying the monitored traffic rate. Analyzing the CPU load results, we can state that running containers in a low-end virtualization add a reasonable low overhead, despite the packet rate increases. The most important result comes out from the comparison of the two NFV based solution: we can run StreaMon probes in Containers with the introduction of a total overhead about 13.47% respect of process solution. Given the limited amount of the overhead, we believe that the solution we designed are promising solutions

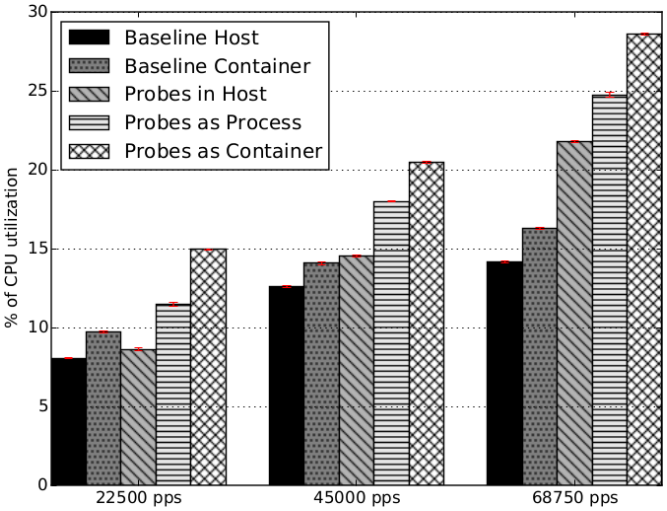


Fig. 5: Cpu Usage

for the targeted environments: service providers can deliver with the same infrastructure (consolidation) both IaaS and the network monitoring without the need for a separated infrastructure of middlebox nodes (different management tools, low level of automation and so on).

V. Related Work

Several monitoring platforms have targeted applications programmability: CoralReef [18], FLAME [19] and Blockmon [20] are frameworks which grant full programmability by permitting the monitoring application developers to hook their custom C/C++/Perl traffic analysis function to the platform. Opensketch [21] proposes an efficient generic data plane based on programmable metric sketches. If on the one hand StreaMon share with Opensketch the same measurement approach, on the other hand its data plane abstraction delegates any decision stage and logic adaptation the control plane and, with reference to our proposed abstraction, does not go beyond the functionalities of our proposed Measurement Subsystem. On the same line, ProgME [22] is a programmable measurement framework which revolves around the extended and more scalable notion of dynamic flowset composition, for which it provides a novel functional language. Even though equivalent dynamic tracking strategies might be deployed over Openflow based monitoring tools, by exploiting multiple tables, metadata and by delegating monitoring intelligence to external controllers, this approach would require to fully develop the specific application logic and to forward all packets to an external controller, (like in Openflow based monitoring tool Fresco [23]), which will increase complexity and affect performance.

Monitoring solutions based on NFV is a topic addressed also by other works. The author of [24] proposes MonPaaS, a framework for resource monitoring of cloud instances that can be used by both cloud providers and consumers. MonPaaS is integrated with Openstack OS and listens to messages exchanged by the Openstack components. Nagios [25] components are automatically configured by MonPaaS and different

strategies of deployments can be realized depending on the scenario. Despite MonPaaS solution share with respect to our work the same target of deploying and manage monitoring probes, it does not take in consideration security aspects like the detection of cyber-threats and it is more focused on the monitoring of the resources. Virtualized functions for security appliances are described in [26]. However this work does not consider tiny virtual resources to run security functions. CloudSec [27] share with our work only the purpose of the real-time security monitoring for the hosted VMs in the IaaS cloud platform. From an architectural point of view is completely different, as it uses monitoring appliances based on introspection techniques. [28] describes a distributed intrusion detection system which spreads its Agents in the Cloud. This agents run in VMs. Also PsycoTrace [29], [30] and other works like [31] exploit virtualization to introduce monitoring solution in the cloud. The approach of running security functions in virtual computing resources is similar to our approach, but they use full-fledged VMs instead of Containers.

VI. Conclusions

This work proposes a distributed NFV framework for network monitoring, which wide-spreads monitoring probes in the infrastructure. By exploiting the proposed architecture, we realized two NFV-based solutions: Processes based and Containers based. We have realized a performance evaluation of the above solutions and, as expected, the amount of the overhead introduced by the NFV designs is limited, making them a very attractive solution. Cloud providers can deliver with a unique infrastructure (consolidation) both IaaS and the network monitoring without the necessity for the deployment of a separated infrastructure of middlebox nodes (different management tools, low level of automation and so on). This will open also a new interesting scenarios where the network monitoring can be provided to the users as a service (XaaS), which can benefit also of the programmability of platforms like StreaMon.

Future works will address: i) the improvement of the packet processing of the D-StreaMon probes; ii) the realization of a coordinated solution which integrates NFV, SDN and Cloud Computing in one infrastructure. A first step towards the improvement of the packet processing could be the replacement of the *Libpcap* library with a more performant solution. The realization of an integrated SDN, NFV and Cloud infrastructure can guarantee the automated deployment of probes and the immediately reaction if a network probe notifies for malicious actions.

Acknowledgment

This research was partially supported by the EU Commission within the Horizon 2020 program, SCISSOR project grant no 644425.

References

[1] Symantec, "Symantec internet security threat report trends for 2016," *Volume XXI, April*, 2016.

[2] A. Galis *et al.*, "Softwarization of future networks and services-programmable enabled networks as next generation software defined networks," in *IEEE workshop SDN for Future Networks and Services (SDN4FNS)*, 2013.

[3] M. Kind *et al.*, "Softwarization of carrier networks," in *Information Technology. Volume 57, Issue 5, Pages 277284*, 2015.

[4] "ETSI group for Network Function Virtualization." [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv

[5] "Software-defined networking: The new norm for networks." [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv

[6] R. Buyya *et al.*, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[7] G. Bianchi *et al.*, "Streamon: A software-defined monitoring platform," in *Teletraffic Congress (ITC), 2014 26th International*. IEEE, 2014, pp. 1–6.

[8] "D-streamon repository." [Online]. Available: https://github.com/netgroup/dstreamon

[9] N. McKeown *et al.*, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[10] P. L. Ventre *et al.*, "D-streamon: from middlebox to distributed nfv framework for network monitoring," in *Local and Metropolitan Area Networks (LANMAN), 2017 IEEE International Symposium on*. IEEE, 2017.

[11] "Scissor project." [Online]. Available: http://scissor-project.com

[12] C. Tankard, "Advanced persistent threats and how to monitor and deter them," *Network security*, vol. 2011, no. 8, pp. 16–19, 2011.

[13] "Ansible." [Online]. Available: https://www.ansible.com

[14] "ZeroMQ Project." [Online]. Available: http://zeromq.org

[15] "Open vSwitch." [Online]. Available: http://openvswitch.org

[16] "Docker." [Online]. Available: https://www.docker.com

[17] "Kubernetes." [Online]. Available: http://kubernetes.io

[18] K. Keys *et al.*, "The architecture of coralreef: an internet traffic monitoring software suite," in *In PAM*. Citeseer, 2001.

[19] K. Anagnostakis *et al.*, "Open packet monitoring on flame: Safety, performance, and applications," in *Active Networks*. Springer, 2002, pp. 120–131.

[20] F. Huici *et al.*, "Blockmon: A high-performance composable network traffic measurement system," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 79–80, 2012.

[21] M. Yu *et al.*, "Software defined traffic measurement with opensketch," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013, pp. 29–42.

[22] L. Yuan *et al.*, "Progme: towards programmable network measurement," *IEEE/ACM Transactions on Networking (TON)*, vol. 19, no. 1, pp. 115–128, 2011.

[23] S. Shin *et al.*, "Fresco: Modular composable security services for software-defined networks." in *NDSS*, 2013.

[24] J. M. A. Calero *et al.*, "Monpaas: an adaptive monitoring platformas a service for cloud computing infrastructures and services," *IEEE Transactions on Services Computing*, vol. 8, no. 1, pp. 65–78, 2015.

[25] "Nagios." [Online]. Available: http://www.nagios.org/

[26] D. Basak *et al.*, "Virtualizing networking and security in the cloud," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 4, pp. 86–94, 2010.

[27] A. Ibrahim *et al.*, "Cloudsec: a security monitoring appliance for virtual machines in the iaas cloud model," in *Network and System Security (NSS), 2011 5th International Conference on*. IEEE, 2011, pp. 113–120.

[28] A. V. Dastjerdi *et al.*, "Distributed intrusion detection in clouds using mobile agents," in *Advanced Engineering Computing and Applications in Sciences, 2009. ADVCOMP '09. Third International Conference on*, Oct 2009, pp. 175–180.

[29] F. Baiardi *et al.*, "Transparent process monitoring in a virtual environment," *Electronic Notes in Theoretical Computer Science*, vol. 236, pp. 85–100, 2009.

[30] B. Payne *et al.*, "Lares: An architecture for secure active monitoring using virtualization," in *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE, 2008, pp. 233–247.

[31] T.-c. Chiueh *et al.*, "Stealthy deployment and execution of in-guest kernel agents," in *In BlackHat*. Citeseer, 2009.