Monitoring and debugging using an SDK for NFV-powered telecom applications

Steven Van Rossem*, Wouter Tavernier*, Manuel Peuster †, Didier Colle*,
Mario Pickavet* and Piet Demeester*
*Ghent University iMinds, Department of Information Technology.
Email: {steven.vanrossem, wouter.tavernier, didier.colle, mario.pickavet, piet.demeester} @intec.ugent.be
† Paderborn University
Email: {manuel.peuster}@uni-paderborn.de

***Abstract*—The next generation of telecom services will require an unprecedented level of flexibility and functionality. In this 5G network context, Software-Defined-Networking (SDN) and Network Function Virtualization (NFV) offer new ways of managing the network and the connected traffic processing nodes. This leads to a programmable way to do Service Function Chaining (SFC), effectively automating the deployment and management of such network services. The growing interest and development in the field of virtualization techniques such as virtual machines or containers further proves that software can indeed play an increasingly important role within telecom services, an area previously dominated by hardware appliances. As a side effect, these softwarization and virtualization techniques also enable a very high degree of customization and configurability, which can also introduce bugs into the service definition, complicating and potentially slowing down service provisioning. In order to close the gap between the development and operations side of a provided service (DevOps), adequate tools are required that allow rapid testing and verification of any modified parameter. To this end we present a set of monitoring and debugging functionalities in the context of aSoftware Development Kit (SDK) that allows a service developer to rapidly setup and provision a network service. By demonstrating how several example services can be deployed in this SDK, we show how this light-weight toolchain can be used to rapidly develop and debug NFV-based services.**

***Index Terms*—SDK, DevOps, SFC, NFV**

I. Introduction and motivation

We start from the viewpoint that a network service consists of several packet processing nodes called Virtualized Network Functions (VNFs). In production, these softwarized network functions can be deployed in multiple datacenters but prior to this stage, it would be beneficial if these VNFs can be tested in a more light-weight environment. Not only to ensure the correct configuration of the packet processing software, but also taking into account additional service related processes such as orchestration, scaling and state migration.

When applying Network Function Virtualization (NFV) to a telecom network service, it consists of several packet processing nodes called Virtualized Network Functions (VNFs). In a production context, these softwarized network functions can be deployed in multiple datacenters but prior to this stage, it would be beneficial if these VNFs can be tested in a more confined, light-weight environment. This would allow testing the configuration the packet processing software, the chaining of functionality, but also taking into account the impact of orchestration, scaling and state migration. Directly testing this in a production-alike environment might involve risks and allow few options for monitoring and debugging. Service providers would benefit from a more light-weight sandbox for NFV (micro-)services, as it enables fast prototyping of new telecom services, combined with a set of tools to check and debug its configuration and functionality. This also implies that the development of such NFV-based telecom services must be compatible with the platform providing them. Network service Management and Orchestration platforms (MANO), typically only provide the operational aspect of deploying a service, but lack specific support for developing, testing and debugging. Frameworks presented in [1] and [2] highlight the need for specialized tools related to NFV-based services but only focus on a single aspect or metric. Tools as developed in [3] try to combine several monitoring functionalities, but miss the ability to emulate or deploy a service for testing. The SONATA project [1] aims to fully incorporate the development aspect as well. A Software Development Kit (SDK) is introduced, allowing to design, test and update services before they are (re-)deployed on a production environment, like SONATAs service platform. This continuous cycle between development and operations, a concept known as DevOps, is a well-known practice from software development. SONATA is developing an NFV framework that provides a programming model and development toolchain for virtualized services, fully integrated with a DevOps-enabled service platform and orchestration system.

It is worth mentioning that all software and examples shown in this demo are open-source and can be freely downloaded from the projects GitHub repository [4] and webpage [5].

II. An SDK for NFV-based services

The SDK is built as a set of small independent tools which can be combined into a workflow to develop a service. This design enables an agile way of working, involving quick and iterative cycles of development, with the possibility to rapidly transition between development and operations, which is one of the key characteristics of the SONATA approach. The software design of the SDK tries to re-use existing

[1] http://sonata-nfv.eu/

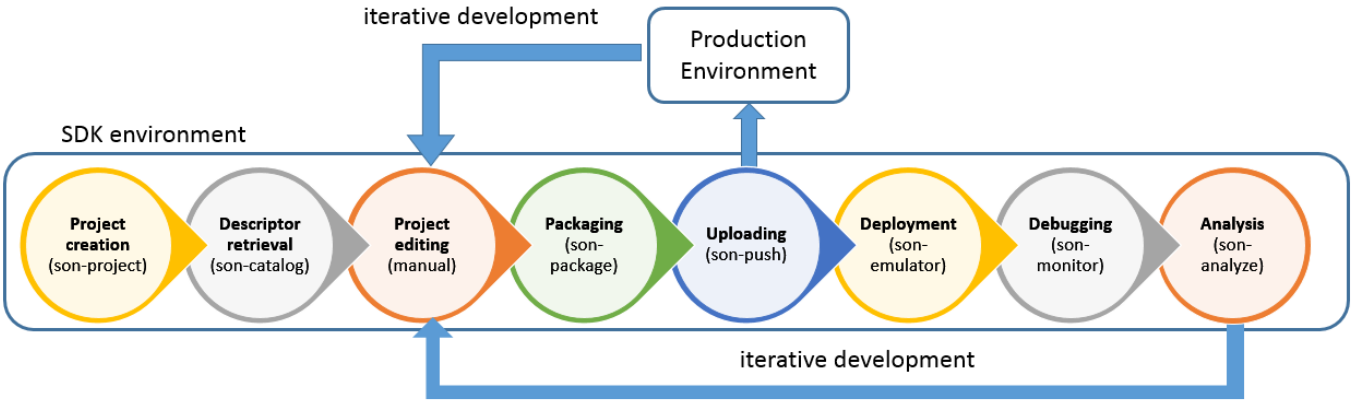

Fig. 1. The SDK flow of development and debugging, with a loop to re-iterate the development process either from the SDK or the production environment.

workflows and concepts in software development such as the use of workspaces, project folders and packaging techniques. This will enable new developers to get acquainted with the SONATA development methodology and tool-set in a short time. Where possible, existing software or libraries have been re-used, to increase the robustness and overall feature set of the SONATA SDK.

Figure 1 shows the global workflow and functionalities implemented in the SDK. The different functionalities implemented in the SDK are listed. They range from tools like editors to describe the service in a specified format, to the deployment, debugging and analysis of the service inside the SDK environment. From the SDK, the service can be pushed to the actual production environment, once development is considered ready. Different re-iterations from both the SDK or the production environment can be done to debug and update a service.

A. Development & debugging process

From the global workflow in Figure 1, we highlight the following two functions in this demo as they form the core of the development and debugging process in the SDK:

- **son-emu**: Extends a Mininet-based environment where VNFs implemented as Docker [6] containers can be deployed and chained, as described in [7] and [8].
- **son-monitor**: A collection of tools that gather and store several monitored metrics, giving the developer a quick and user-defined real-time overview of those metrics, helping to test and debug the service. This is further detailed in section II-C.

As can be seen in Figure 2, the emulator is able to :

- Emulate an available infrastructure as a network of linked datacenters or Points of Presence (PoP), interconnected by switches. (Two PoPs in this case)
- Deploy VNFs (in the form of Docker containers) inside the PoPs at runtime.
- Chain the different VNFs together.

Likewise, as depicted in Figure 2, the monitoring framework is able to :

- Monitor compute, storage and network metrics (cpu, mem, traffic rates) of the VNFs themselves.
- Monitor custom and user-defined flows inside the emulator network (inspect the links between the VNFs).
- Start specific commands in the access points of the service to measure end-to-end metrics (eg. start a packet stream at the input and monitor the delay and packet loss at the output).

These tools form the base for a sandbox that can be used to easily deploy and monitor NFV-based services. The monitored metrics and provided functionality allow a developer to check the configuration and basic performance of each VNF and the total service.

B. Service components

The SONATA programming model focuses on the common concepts of service chains and service graphs comprised by individual network functions. Each of these components are defined by their corresponding descriptors following a particular data model. The used data models are built upon ongoing standards (ie. TOSCA or ETSI) or outcomes of research projects (ie. FP7 T-NOVA, UNIFY), extending them when appropriate. A SONATA service is characterized by a network service descriptor (NSD), a set of virtual network function descriptors (VNFD), and a package descriptor for the overall service. Each of these descriptors define artifacts containing data and information such as images, files and/or configuration parameters or scripts. The descriptors follow a YAML schema language format. A set of validation tools have been implemented to check if given descriptors are compliant with the SONATA format.

Also visible in Figure 2 are the inputs to the emulator and monitoring framework. The NSD and VNFD descriptor files include the necessary information to deploy the VNFs and the links of the service in the emulator. The list of monitored metrics can also be parsed from a descriptor file or manually given via the command-line. The VNFDs and NSD are packaged and sent to the emulator for deployment.

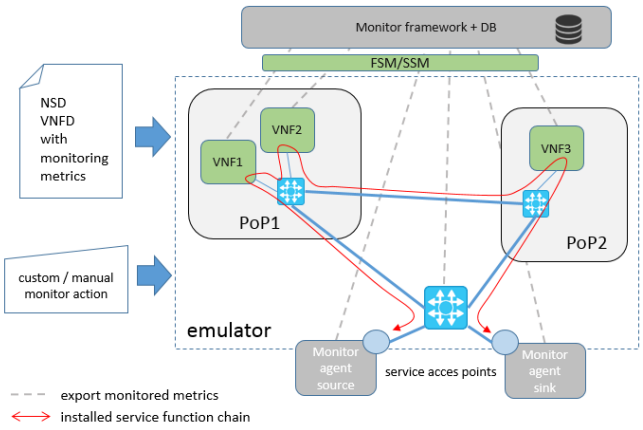

Fig. 2. High-level architecture of the SDK showing the emulator and its monitor framework including a deployed service.

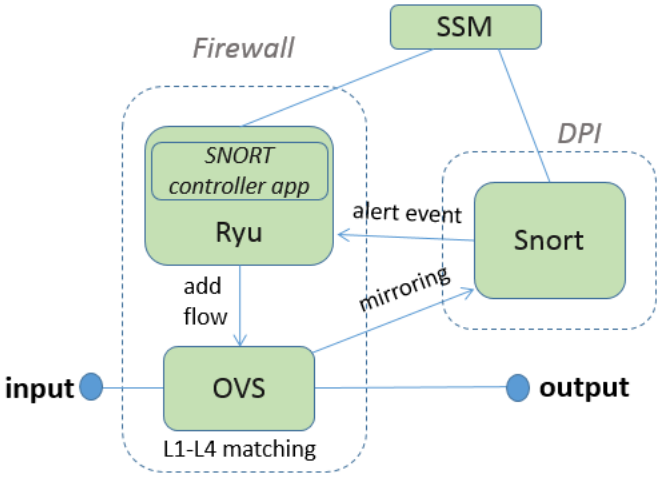

Fig. 4. An IDS service consisting of multiple VNFs and an SSM.

A specific architectural aspect of the SONATA platform is that the service developer can ship the service package to the service platform together with service- or function-specific management or configuration code, expressing and realizing requirements and preferences. This code is referred to in SONATA as Service-Specific Managers (SSM) and Function-Specific Managers (FSM), respectively. SSMs and FSMs can influence the Service and VNF lifecycle management operations, e.g., by specifying desired placement or scaling behaviour. This grants the developers increased flexibility, control and resilience of their service. The architecture of the SONATA service platform is documented in [9]. Also in a context of closed-source VNFs, where a developer cannot make internal modifications inside the VNF code, the SSM might offer a way to translate or 'glue' the API of one VNF to that of another one. This is illustrated in Figure 2 where an SSM is connected to the management interface of different VNFs in order to configure them and receive additional data from them. The SDK can be a sandbox to test this SSM and FSM code since the interfaces to the VNFs are available when deployed in the SDK. The SSM/FSM will use the VNF APIs to communicate with the VNFs and it can be tested if these APIs are indeed available and working as expected.

C. SDK architecture

A detailed breakdown of the SDK architecture can be seen in Figure 3. **Son-monitor** is part of SONATA's son-cli repository and can be installed on the development machine from a debian package or as a docker container. **Son-emu** can be deployed as a Docker container (requiring privileged root access to control the networking of the SDK machine) or as an isolated VM using a Vagrant script.

All installation instructions can be found in the SONATA github page: [4]. The SONATA specific Docker containers are also available from its public Docker hub[2].

[2]https://hub.docker.com/u/sonatanfv/

Additional containers are being deployed to help gather, store and visualize monitored metrics:

- cAdvisor [10] : Analyzes resource usage and performance characteristics of running containers.
- Prometheus DB and pushgateway [11] : Used as database to gather and temporarily store the monitored metrics.
- Grafana [12] : Visualization of the metrics.

A complete description of the SDK and its current capabilities can be found in [13].

One of the main advantages of this architecture is that it's light-weight enough to be executed on a single machine (eg. the service developer's laptop) and allows rapid prototyping of NFV-based services.

III. Demonstration: Developing and Debugging an Intrusion Detection Service

To illustrate the capabilities of this SDK, we want to deploy the service illustrated in Figure 4, by sending the appropriate service package to the emulator. By generating traffic and monitoring specific link and VNF metrics, we will be able to identify possible bugs or a misconfiguration in the service. Once verified and assumed to be correctly functioning in the SDK, the service package is ready for the production environment.

A. Service description

This Intrusion Detection System (IDS) consists out of a Firewall (Ryu Openflow controller + OpenVSwitch) and a Deep Packet Inspector (SNORT). The Firewall mirrors every received packet on a port to the DPI. This mirrored traffic is analyzed and when malicious traffic is found, the DPI sends an alert to the Ryu controller of the Firewall, instructing it to block certain flows of traffic. This is translated to installing a blocking flowrule in the OpenVswitch's flowtable, effectively blocking the malicious traffic.

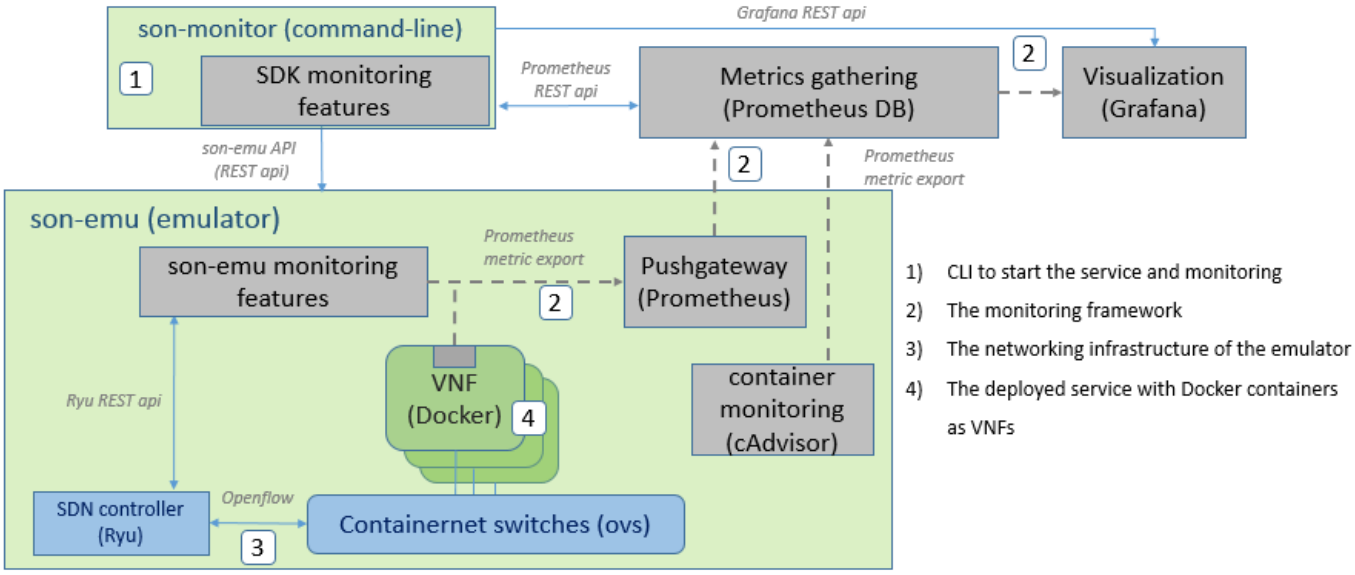

Fig. 3. Low-level architecture of the SDK showing the emulator and its monitor framework, including the different sub-components.

B. Components and descriptors

The NSD and VNFD yaml files that describe the service can be shown during the demo and are sent as a package to the emulator to be deployed. They basically inform the emulator to deploy every VNF and link shown in Figure 4. Additionally we construct a monitoring descriptor file that contains all the metrics we want to monitor while the service is running in the SDK. This monitoring descriptor file is parsed by the son-monitor command and as a result a Grafana dashboard where all the metrics are shown in real-time is automatically created. Figure 5 and 6 illustrate the Grafana output. This dashboard offers the main overview for the developer to check if the service is functioning correctly.

Fig. 5. VNF metrics (cpu, mem, traffic rate) monitored by Grafana

C. Debugging

By using a combination of user-defined metrics and generated traffic from the service access points, it will be possible

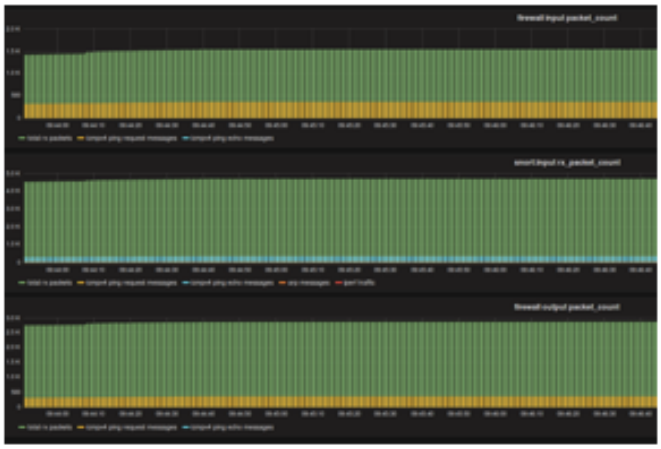

Fig. 6. Packet counters for user flows monitored by Grafana

to verify the correct functionality of the service. We assume the service developer has pre-installed some rules in SNORT to detect and block traffic such as ping flood attacks or a website that should be blocked. By generating ping messages and website requests from the service access points, we can monitor if and how long it takes for the service to effectively block the traffic. By monitoring cpu rate, traffic rate and end-to-end packet loss, we can get a first idea of the service performance while generating different packet streams. Different packet generators can be used such as iperf, netperf, scapy or tcpreplay to stream a pre-created pcap file. In addition to checking the service functionality, the monitoring framework can assist in debugging specific bugs or errors that might occur when creating NFV-based network services.

1) NSD fault: This fault is caused by an error is the NSD file. This can be a link between two VNFs that is wrongly

defined in the NSD. For example if the link between OVS and Ryu in Figure 4 is forgotten or attached to another interface, we will not monitor OpenFlow tcp packets on the link of ovs to Ryu. By monitoring the packet rates on any VNF interface a broken link can be easily identified.

2) VNF fault: This can be caused by a misconfiguration of the VNF or a VNF software bug. For example if the OVS is wrongly configured and sends the mirrorred traffic out on another port, then SNORT will not receive any traffic to analyze and no alert will be sent. This will be noticed when monitoring the mirrored traffic and alerts on the specified links.

3) SSM fault: This kind of error is related to failing service specific management, for example the wrong use of the API of a VNF. If for example the user wants to install a new rule into SNORT this can be automated by letting the SSM ssh into the VNF and modify the configuration files or rule files. Also the REST API of Ryu can be used to configure additional rules in the firewall. In the SDK, we can test the any API call the SSM would need to use.

IV. Future work

The SDK emulator is currently limited to deploying Docker containers. These are relatively light-weight VNFs that enable a local deployment on a single machine. Some production environments lack however good support to deploy containers and only accept Virtual Machines (VMs). To keep the SDK light-weight and fast, it is not recommended to deploy actual VMs inside the emulator, but it can be investigated how a Docker container can be automatically transferred to a VM and how much the performance overhead can be minimized.

Building specialized monitoring agents that can be deployed as VNFs inside the service enable monitoring features that can be moved from the service endpoints to anywhere inside the service. Our research will further build upon this to develop standard validation tests for SFC's and profiling procedures that help in the prediction of VNF performance.

V. Demo related topics

To show the demo, only the presenter's laptop will be used. One additional monitor (to be provided by the organization) is recommended. As tentative schedule, minimal 5 mins. would be necessary to explain the basic architecture and show a running example. This includes a walk-through of the development cycle starting from the creation of a network service (including introduced bugs), its deployment in the emulator of the SDK, demonstration of monitored metrics, debugging actions, and redeployment.

VI. conclusion

The presented SDK environment is a light-weight sandbox that allows rapid prototyping of NFV-based services. The SDK provides the developer a confined environment, where network services can be evaluated in terms of functionality rather than pure performance, monitored and debugged before they actually need to reach the market or production environment. This implementation has the big advantage that service chains can be set up, tested and monitored in a very short timeframe (seconds to minutes), allowing fast reboots and reconfigurations of parameters inside the VNFs or in the emulator network. After the NFV-based service has been tested in the SDK, a basic verification of its configuration and descriptors has been done, increasing its chances of a successful deployment in the production environment.

All software and examples shown in this demo are open-source and can be freely downloaded from the project's GitHub repository [4] and webpage [5]. More example services and VNFs will be included in the SONATA GitHub pages as the project continues.

Acknowledgement

This work has been performed in the framework of the SONATA project, funded by the European Commission under Grant number 671517 through the Horizon 2020 and 5G-PPP programs. The authors would like to acknowledge the contributions of their colleagues of the SONATA partner consortium (www.sonata-nfv.eu).

References

[1] H. Mahkonen, R. Manghirmalani, M. Shirazipour, M. Xia, and A. Takacs, "Elastic network monitoring with virtual probes," in *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on*. IEEE, 2015, pp. 1–3.

[2] M. Xia, M. Shirazipour, H. Mahkonen, R. Manghirmalani, and A. Takacs, "Resource optimization for service chain monitoring in software-defined networks," in *2015 Fourth European Workshop on Software Defined Networks*. IEEE, 2015, pp. 91–96.

[3] I. Pelle, T. Lévai, F. Németh, and A. Gulyás, "One tool to rule them all: a modular troubleshooting framework for sdn (and other) networks," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. ACM, 2015, p. 24.

[4] SONATA consortium, "SONATA github," https://github.com/sonata-nfv, 2016.

[5] SONATA project, "SONATA webpage," http://sonata-nfv.eu, 2016.

[6] Docker Inc., "Docker: An open platform for distributed applications," Website, August 2013, online at http://www.docker.com/.

[7] M. Peuster, H. Karl, and S. van Rossem, "Medicine: Rapid prototyping of production-ready network services in multi-pop environments," *CoRR*, vol. abs/1606.05995, 2016. [Online]. Available: http://arxiv.org/abs/1606.05995

[8] M. Peuster, "containernet," https://github.com/mpeuster/containernet, 2016.

[9] SONATA consortium, "D2.2 Architecture Design," http://sonata-nfv.eu/content/architecture, 2015.

[10] Google, "cAdvisor," 2014. [Online]. Available: https://github.com/google/cadvisor

[11] Prometheus authors, "Prometheus - Monitoring system and time series database," 2016. [Online]. Available: https://prometheus.io

[12] Grafana, "Grafana - Beautiful metric and analytic dashboards," 2015. [Online]. Available: http://grafana.org/

[13] W. Tavernier, S. V. Rossem, T. Batista, M. Bredel, G. Chollon, M. S. Siddiqui, M. Peuster, and S. Draxler, "Deliverable 3.1: Basic SDK Prototype," http://sonata-nfv.eu/content/d31-basic-sdk-prototype, SONATA consortium, Tech. Rep., 2016.