Empowering Network Service Developers: Enhanced NFV DevOps and Programmable MANO

Thomas Soenen, Wouter Tavernier, Manuel Peuster, Felipe Vicens, George Xilouris, Stavros Kolometsos, Michail-Alexandros Kourtis and Didier Colle

***Abstract*—Although network function virtualization redefined the role of the network service developer, existing concepts that are supposed to enable them are still limited, cumbersome and time consuming in regard to the promised flexibility. This article describes how to move forward from these initial steps, identifies the challenges network service developers face both during development and runtime, and explains how to overcome them with our service construction kit (SCK) and programmable MANO framework. We detail how our SCK facilitates the service creation process, resulting in two enhanced NFV DevOps cycles, test flexibility and quicker service development. We elaborate on our programmable MANO framework with both architectural considerations and a use case, to depict its customisability by network service developers, giving them finegrained control over their service throughout its end-to-end operational lifecycle.**

I. Introduction

The rise of network function virtualization (NFV) has broken the lock-in between hardware and software for network functionality. Before, providing a new network function meant designing and creating one or more physical middleboxes, often in narrow cooperation with telecom operators, which had to be placed in the network. Now, one only needs to provide the software, typically packaged as a virtual machine or container, and the telecom operator executes this virtual network function (VNF) on general-purpose hardware at one of its premises. In theory, this transition enables every programmer to become an NFV developer, calling for a re-assessment of the developer roles. We identify two distinct roles:

- The **VNF developer** develops single VNFs. Their responsibility starts and stops at the VNF interfaces, so they aren't concerned with end-to-end metrics. They can be considered as application developers which are unaffiliated with telecom operators.
- The **network service developer**, further also referenced as the developer, creates end-to-end services by chaining one or more VNFs together. To ensure that the service provides the required QoS for its users, the developer is concerned about the functionality offered by the selected VNFs and about capabilities such as latency and throughput of the network infrastructure used to connect them. The developer researches which VNFs to use, designs the service function chain (SFC) and develops and tests the associated descriptors, configuration files and code. These artefacts describe how to (re)configure the service and its components for various regimes they might operate in. This developer needs a deeper knowledge about network infrastructure and their providers, but isn't necessarily affiliated with them.

In this article, we contribute to the state-of-the-art of the NFV domain by empowering these developers in quickly developing products that provide the required stability in every regime they might operate in.

The first contribution of this article is a modular programmable management and orchestration (MANO) framework that can be customised per network service or VNF. The MANO framework extends its functionality with workflows created by the network service or VNF developer. These workflows — we refer to them as service and function specific managers — replace the generic MANO behaviour only for that specific network service or VNF. This grants developers the control they require over the operational lifecycle, since only they know how to manage and configure their product in different regimes. We explain the architecture and exemplify the mechanism by using it to add non-generic self adaptation capabilities to a rudimentary content delivery network.

The second contribution is a service construction kit (SCK), a set of tools that ease the life of the developer. Most importantly, it includes an Emulator which allows a developer to emulate a telecom operator's infrastructure of virtualised compute and networking resources on a local machine. This allows the developer to perform large amounts of testing effort locally for various topologies, making for shorter development cycles. Final testing still needs to occur on operational NFV infrastructure. For this, the SCK offers tools that interface with such environments to upload and instantiate the service to be tested, to configure and trigger troubleshooting engines that will generate different regimes for the service to operate in and to extract monitoring data in order to evaluate the service's performance. We argue that these tools, combined with the Emulator, significantly enhance the NFV DevOps cycle and thus greatly improve developer's flexibility to evolve network services quickly and incremental. As a final contribution, we present an assessment of this DevOps model.

II. Service and Function Specific Management and Orchestration

A network service might experience various regimes during its runtime lifecycle. For example, a change in user behaviour can create a tangible increase in its load, or the performance might degrade due to saturation of the used infrastructure. To

Thomas Soenen, Wouter Tavernier and Didier Colle are with Ghent University - imec; Manuel Peuster is with Paderborn University; Felipe Vicens is with Atos Research & Innovation; George Xilouris, Stavros Kolometsos and Michail-Alexandros Kourtis are with NSCR Demokritos.

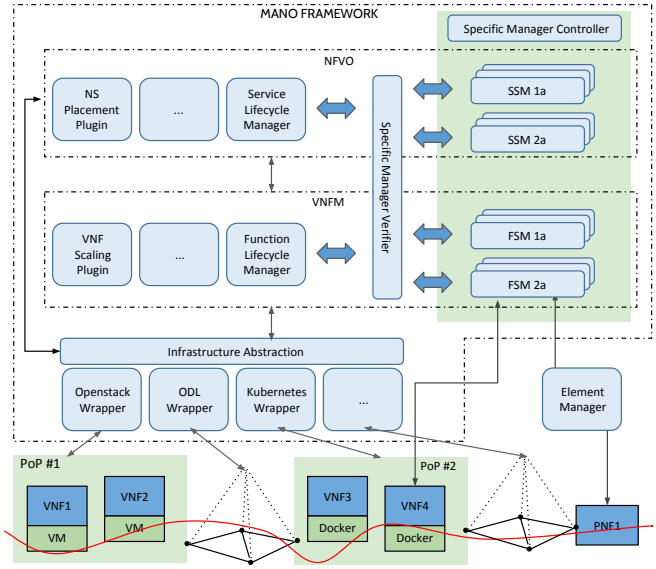

Fig. 1: MANO framework Architecture with Specific Managers.

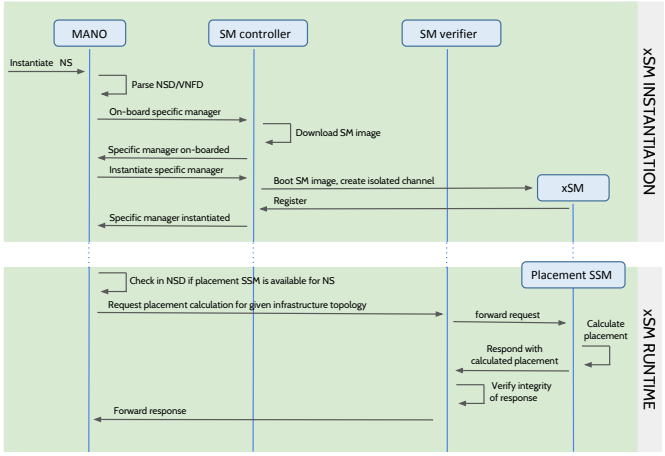

Fig. 2: Specific Manager instantiation and runtime sequence diagram.

ensure end-to-end QoS offered by a network service during its lifetime under various regimes, they need to be flexibly reconfigurable to adapt upon any given situation. In many cases, it is solely the developer that knows which lifecycle events (e.g. how to scale the service, how to reconfigure the SFC, which VNF to migrate) assure the required QoS when new regimes occurs. A similar scenario arises when instantiating a new network service: the location where VNFs are deployed can have a significant impact on the QoS. Again, only the developer knows which placement decisions for a given resource availability and topology yield the intended objective. For this reason, a MANO framework should be programmable by the developer, to make network service or VNF lifecycle events consider developer input and to trigger developer defined lifecycle events based on monitoring data.

One solution could be that developers add this specific knowledge to the network service and VNF descriptors. This allows the MANO framework to remain generic — all service and VNF instances are subject to the same workflows — but limits its flexibility and potential. Describing a placement algorithm through descriptor language will either be highly complicated or limit the options of the developer to a predefined set. Reconfiguration payloads, and how to inject them, for running VNFs can be defined in numerous different ways, making it impossible to support all of them through descriptor language. Keeping the MANO generic leads to suboptimal services, cutting into the potential of the NFV ecosystem. Therefore, it is our believe that MANO framework workflows should be customisable per network service and VNF by the developer.

Looking at the state-of-the-art, some MANO customisation features were proposed. OSM [1] and Open Baton [2] allow developer-defined function specific VNF managers (VNFM) — implemented as Juju Charms — to be attached to their frameworks. Both platforms instruct these Juju Charms instead of the generic VNFM for lifecycle events (e.g. starting, configuring or healing a VNF) of associated VNFs. Both platforms miss the ability to customise lifecycle events on the service level — the VNFM operates only on VNFs — which are implemented in the NFV orchestrator (NFVO). If (part of) a network service requires migration, a generic NFVO doesn't know how to calculate the new placement while guaranteeing acceptable QoS. Similar for scaling a network service; a generic NFVO doesn't know which additional VNFs to deploy, which existing instances to reconfigure and how to connect them in a new SFC. These events can only be supported if the NFVO is customisable as well. ONAP introduced a concept for a closed loop automation management platform (CLAMP) [3] which allows operators to create a control loop with access to monitoring data. Required lifecycle events resulting from diagnoses are described by policies, which can instruct the NFVO. Although this allows operators to customise the NFVO to some extend, it misses fine-grained control and can't be used by developers. From academic work, [4] and [5] mention the necessity for NFVO and VNFM customisation without proposing solutions.

We solve this necessity by introducing a MANO framework that maximises the programmability by developers. Our mechanism — first mentioned in [6] and shown on Fig. 1 — allows service specific managers (SSM) and function specific managers (FSM) to be attached to the MANO framework. Specific managers are processes created by network service and VNF developers that incorporate service and function specific MANO behaviour. The network service or VNF descriptor indicates where they are stored and which lifecycle events they customise. Once the associated network service or VNF is instantiated by the MANO framework, they are onboarded, instantiated and attached to the framework as shown in the xSM instantiation block on Fig. 2. For each lifecycle event that needs to be executed, the MANO framework looks up in the associated descriptor whether a specific manager exists that customises this event. If that is the case, this specific manager will be inquired to execute that workflow. If not, the generic MANO framework workflow is executed. Among

others, an SSM can provide custom workflows that describe the placement calculation during instantiation, how to scale the network service, how to heal it, reconfigure it or how to react to associated monitoring alerts. In terms of customising the VNFM, an FSM can define how to scale a VNF, how to configure it after instantiation or migration, how to heal it, etc. Of course, each specific manager has to consume the correct MANO framework API [7]. In terms of customising VNF lifecycle events, our mechanism is similar to those in [1] and [2]. The main contribution steams from the SSMs, which give network service developers a fine-grained control over the NFVO, a feature unavailable in other MANO frameworks.

For example, when instantiating a new network service, the mapping of the VNFs and virtual links onto the substrate needs to be calculated. Since only the developer knows, given the infrastructure topology and its capabilities, which mapping results in acceptable end-to-end network service QoS, he or she can embed the placement algorithm that implements that knowledge in a placement SSM. The MANO framework knows from the descriptor that a placement SSM is available for this service, and will invoke it when a mapping is needed, instead of using the generic placement algorithm. This call to the SSM includes a view of the available infrastructure, and the MANO framework verifies whether the response yields a realistic mapping. This process is shown in the xSM runtime block on Fig. 2. More complex structures can be created. An SSM can program how the MANO framework reacts to monitoring alerts. The MANO framework forwards all the monitoring data associated to the network service to this SSM. This allows the SSM to diagnose suboptimal performance of the network service and undertake steps to resolve this by instructing the MANO to execute a custom workflow. Among others, such a workflow can contain instructions to migrate (which first require an invocation of the placement SSM), scale (such as adding VNFs and reconfiguring the SFC), heal (e.g. by triggering an FSM that accesses the VNF and restarts a process) or a combination of the above.

To ensure that the developer doesn't abuse the granted freedom to undermine the MANO integrity, some measures need to be taken. Communication between the MANO framework and any specific manager is isolated, protecting sensitive control information of other network services. Messages from specific managers are screened by a specific manager verifier which incorporates a set of verification tools, such as [8]. These evaluate whether requested configurations are feasible, whether proposed workflows make sense (e.g. chaining instructions can never precede placement calculations), etc. A specific manager controller is required to manage their lifecycles (e.g. on-boarding, instantiating and registration) and to monitor there resource usage. Interactions with the specific manager controller and verifier are detailed on Fig. 2.

An operational implementation of this MANO framework design was created during the SONATA and 5GTANGO projects, and is shown on Fig. 1. The generic NFVO behaviour is implemented by the service lifecycle manager and the function lifecycle manager implements the generic VNFM, both using a set of submodules. xSMs are executable binaries. To aid the developer, the implementation provides an xSM source code template that allows it to connect and register with the MANO framework. The template's main function consumes messages from the MANO framework and, depending on their content, triggers specific callback functions. By defining the logic in these callback functions, the developer customises the MANO. For example, if an SSM receives a request to evaluate monitoring data, the associated callback contains the developer's code that analyses the data and decides whether a scaling event is needed. When the function concludes, the result is sent to the MANO. The developer can use the available template, or start from scratch. In that case, the MANO framework API isn't available through predefined functions and the developer should be careful to obey it. Our implementation comes with a tool to test whether xSMs obey this API and how each callback function behaves for all possible MANO inputs.

III. Specific manager case study: self adaptation for a Content Delivery Network

To exemplify the specific manager mechanism introduced in the previous chapter, we use it to add self adaptation capabilities to a rudimentary content delivery network (CDN). Users from two adjacent access networks request to stream content. The CDN uses instances of two VNFs: a video content cache (vCache) able to provide and store content and a traffic classifier (vTC) that inspects incoming traffic and decides upon the appropriate service chaining. The vTC intercepts all ingress traffic from the network service and the traffic steering decisions implemented at the vTC are:

- Traffic is a user web traffic request, forwarded to vCache
- Traffic is streaming media content coming from the streaming server, forwarded to vCache
- Traffic is streaming media content stemming from vCache, forwarded to the user

When the user makes a request, it is intercepted by the vTC and forwarded to the vCache for processing. If the content isn't available, then the vCache requests the content from the original streaming server where the content is available. In our network topology one micro datacenter is present in each access network, which is considered the optimal location for both VNFs to provide content with high bandwidth and low delay. Each access network has a vTC, where all requests from that access network are consumed through chaining. Since a vCache can potentially consume many resources, only one is deployed and used by both vTCs. The setup is shown on Fig. 3. To lighten the load of the core network, streams originating in the media server are capped at 12 Mb/s, too low according to Netflix to stream Ultra-HD (4K) content. For this scenario, we assume the requested content is available in the vCache.

To ensure high bandwidth, we extend the service with self adaptation functionality for the vCache. We first add a monitoring SSM. We extend the VNFs with a monitoring probe that sends values for custom metrics towards the monitoring framework. The vTC sends a metric that indicates whether it can ping the vCache, the vCache sends an indication of its load. These metrics are defined in the VNF descriptors, so the monitoring framework knows what to expect. The MANO

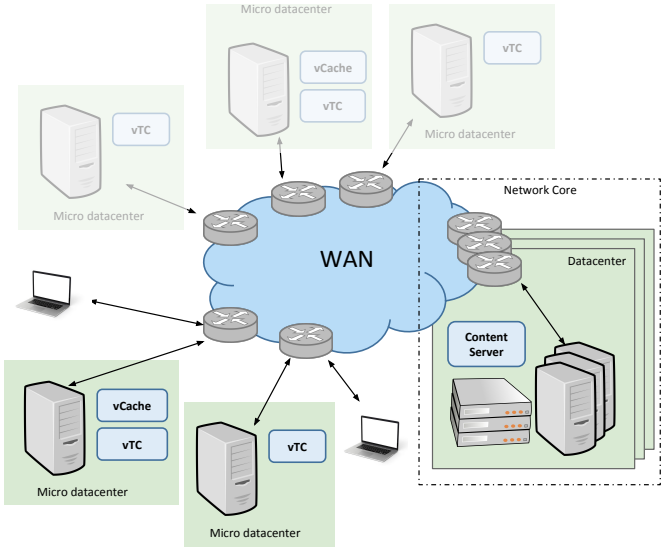

Fig. 3: rudimentary content delivery network.

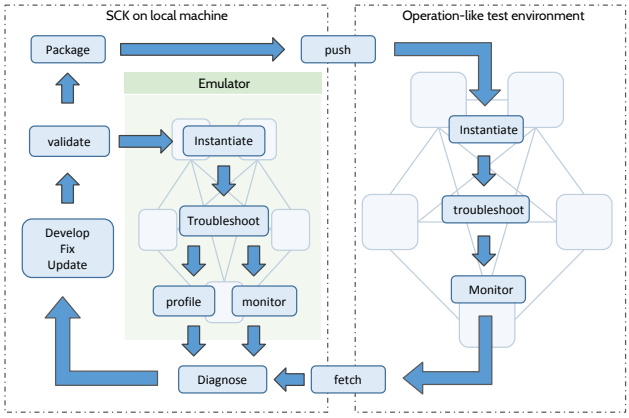

Fig. 4: SCK Workflow for network service development.

forwards all received data for this service coming from the monitoring framework to the monitoring SSM. We also add a healing FSM for the vCache, which is designed to salvage a corrupt vCache instance (e.g. by connecting to the VM, running a diagnostic check and restarting a process). When the monitoring SSM detects that the vCache is no longer available, it instructs the MANO framework to trigger the vCache healing FSM. Whether the FSM succeeds or fails (e.g. the FSM can't connect to the VM because it crashed) is reported to the monitoring SSM. If failed, the monitoring SSM instructs the MANO to execute a custom workflow that terminates the old vCache, instantiates a new one, includes it in a new SFC and triggers the respective vTC configuration FSMs to instruct each vTC to start using the new vCache. Since vTCs direct requests to the media server when no vCache is available, these scenarios see a reduced bandwidth observed by the users.

A longer lived bandwidth drop might not be bridgeable by buffering and could result in degradation of video quality. To prevent this, we extend our SSM to deal with this proactively. We extend the vTC monitoring probe to send the number of users being served. Once this reaches a threshold, the monitoring SSM decides an additional vCache is needed. This threshold is a trade-off between additional resource cost and the improved QoS of users. The monitoring SSM instructs the MANO to execute a workflow that contacts the placement SSM to calculate where to place the vCache, instantiates the new instance, updates the SFC and finally informs the vTCs that two vCaches are available. The placement SSM places the vCache so that the number of users with increased bandwidth is maximised. Each vTC polls the vCaches frequently for their load, to decide which vCache to use. Once a vCache becomes unavailable, both vTCs will use the remaining vCache. As before, the monitoring SSM will try to salvage the unavailable vCache or instantiate a new one.

This workflow describes a non-generic scaling process to facilitate self adaptation. Looking at large scale deployments, more access networks and vCaches will be included and each vTC is only informed with nearby vCaches. To ensure vCache availability, more scaling events might be required if the load approaches their limits. The monitoring SSM has two ways to go about this: either instruct the MANO to deploy additional vCaches or reconfigure vTCs to use those vCaches with a below average load.

IV. Enhancing the NFV DevOps loop with an SCK

DevOps aims to unify software development and software operations to reap synergetic benefits. As per [9] and [10], this is achieved through i) the availability of automated processes to test products under construction repeatedly and reliably, ii) development and testing is done against production-like systems, iii) the availability of continuous monitoring to validate operational quality and iv) development is done in an iterative and incremental manner through short iterations and systematic testing. As argued in [11] and the associated IETF draft [9], existing IT DevOps mechanisms cannot be directly applied to the NFV context due to different requirements and infrastructure scale. Tools that aid development of datacenter applications don't account for the large, heterogeneous and geographically spread telecom infrastructure or for the carrier-grade requirements in terms of availability or latency. [11] proposes an NFV DevOps model with verification, troubleshooting and monitoring tools up to telecom standards. However, all tools are executed by the operator, and thus within an operations environment. The model lacks concepts that aid the developer in their local environment. For simplicity, the authors also assume that service developers and telecom providers are the same entity. Consequently, the model doesn't consider independent service developers that have no access to operation environments.

We use the remainder of this section to present our SCK [12], a toolkit with DevOps tools for both independent and affiliated network service developers. Its key feature is the Emulator [13], a tool that emulates topologies of telecom operator infrastructure — with both virtual compute and networking resources — on a local machine. This enables developers to test large parts of the functionality of their service locally,

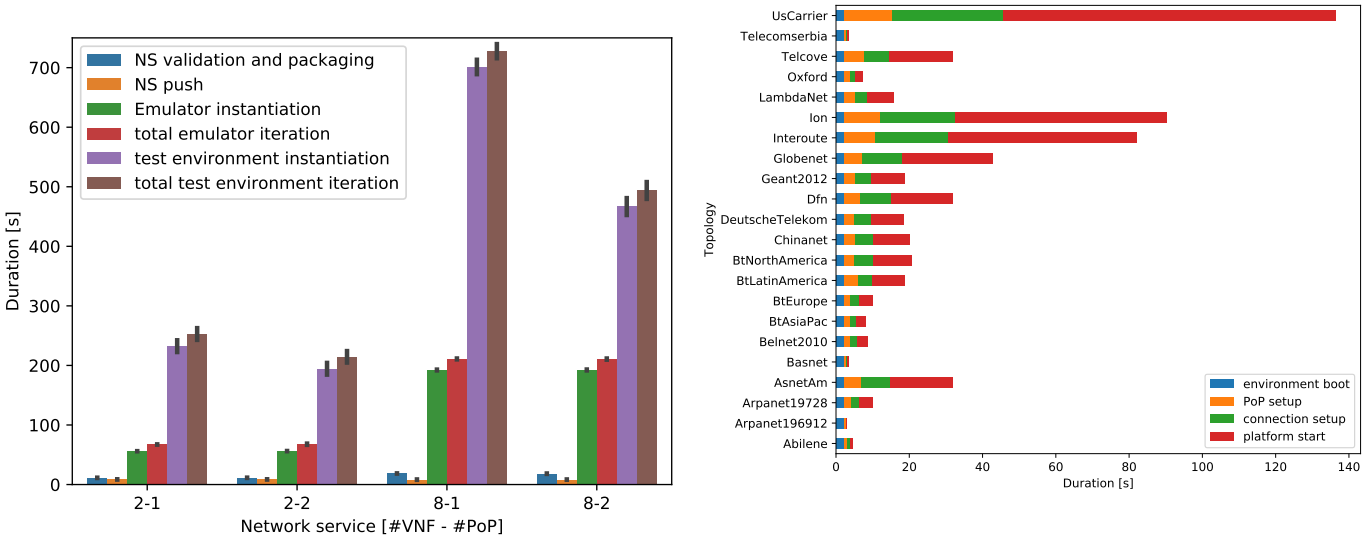

Fig. 5: SCK performance: a) step duration in both development cycles; b) emulator setup time for various infrastructure topologies.

including the SFC and end-to-end behaviour, without needing to interface with a telecom operator's test environment. The Emulator provides troubleshooting features [14] with end-points that generate traffic to simulate different regimes for the network service to operate in. The SCK profiling tool can be used to suggest improved resource requirements or scaling behaviour based on these results. Since resources available in the Emulator are limited and shared with the control plane final testing on production-like test environments is still required. To interface with them, the SCK packages all network service artefacts conform ETSI's SOL004 [15] package specifications, pushes them to the test environment and instantiates them. The SCK extracts monitoring data from these test environments, allowing developers to quickly diagnose any issue.

The SCK workflow is detailed on Fig. 4 and shows two development cycles: one using the Emulator to test the service locally and one using operator infrastructure in a test environment. Once the service stabilises in the local cycle, it is tested in the operations cycle. If any test fails, the network service is updated and the procedure repeats itself. This SCK workflow satisfies all DevOps requirements: i) All processes, except for programming and diagnosing, are completely automated to prevent human error, ii) a production-like environment is locally emulated for rapid testing and interfaces with telco test environments are provided, iii) monitoring data is continuously collected and can be fetched anytime to ensure issues are logged and iv) the average development cycle is substantially shortened, as shown in the section below, allowing for incremental development and fast convergence towards stable network services. Relating to the state-of-the-art, these SCK features further enhance the NFV DevOps loop for every network service developer.

V. Shortening Network Service Development Iterations

Developing a network service requires various steps. The developer needs to select VNFs, describe how to interconnected them, build configuration scripts (i.e. SSMs), etc. Typically, such development requires multiple iterations where intermediate versions are tested and improved. Since a DevOps mindset tends to further increase the number of iterations, minimising their duration is of paramount importance to converge quickly to a stable product. Here, we compare the duration of a single iteration for both types of development cycle that are supported by our SCK and shown on Fig. 4.

The time it takes to code or update service artefacts, or to diagnose an issue in test data depends on the developer's skillset. Since this is difficult to quantify and independent of the selected development cycle, it is omitted from the measurements. Fetching monitoring data is a continuous process and omitted as well. We assume both development cycles use identical troubleshooting tools, making its duration cycle independent. As this duration is difficult to estimate — it depends on how extensive the service is tested — it is also omitted. Therefore, we compare the duration of both cycles based on these SCK steps: validate, package, push, and instantiate. Results are shown on Fig. 5.

The figure shows the duration of the steps for four different network services, two containing two VNFs spread over one or two datacenters and two with eight VNFs, also evenly spread over one or two datacenters. Results are averaged over ten measurements. Every VNF is an Ubuntu Cloud VM operating as a switch. For both cycles, the duration is dominated by the instantiation phase, and it is sensible longer on the test environment, here managed by a SONATA service platform (SP). This is due to the lightweight nature of the Emulator, which only needs to instantiate the service, compared to the SP that performs a lot of additional work that is obsolete for test scenarios (e.g. related to security or communication between

various SP components, storing records, etc.). Additionally, the Emulator uses local resources, while the SP orchestrates on top of remote virtualised resources available from datacenters and networking equipment, typically controlled by virtual infrastructure managers (VIM) like OpenStack and OpenDayLight. Therefore, the SP has to interface with such VIMs to instantiate the service, a much slower process than controlling locally emulated resources. Test environment instantiations are faster when two datacenters are involved, since some work is done in parallel.

Figure 5 shows instantiation times on a simple telco infrastructure with one or two datacenters. Our experiments show that it takes the Emulator roughly two seconds to emulate this type of topology. Figure 5 shows how long it takes the Emulator to set up more realistic telco topologies, obtained from *The Internet Topology Zoo*. It takes a couple of minutes for a few of them, but most are emulated in less than a minute. Within a matter of minutes, the Emulator allows a developer to test a network service on a large and diverse set of realistic telco infrastructures, something that is close to impossible when having to interface with each operator's environment directly to test on their topology.

The experiments show that the Emulator-based development cycle is faster and far more flexible in terms of tested topologies than the test environment cycle. Introducing the Emulator clearly reduces overall development time of network services, especially in a DevOps scenario where the number of iterations increases. As a best practice, we suggest that developers use the Emulator development cycle up to the point where the service performs as desired for targeted telco topologies. Then, the test environment cycle should be used to validate that the service performs similarly on targeted production-like environments. As a final note, we compared the efficiency of our SCK with those from OSM and ONAP. For each SCK, we modelled the number of developer steps required for a single cycle from programming or updating network service artefacts until on-boarding the package. For each SCK, we observed a linear correlation between this number of steps and the number of VNFs in the service. The slope of this correlation is more or less twice as steep for the other SCKs when compared to ours. This can be explained by the high level of integration between our tools resulting in a higher degree of automation, and by our service model where services and their VNFs are packaged as a single package, greatly simplifying on-boarding procedures.

VI. Summary

The role of network service developer has evolved with to the introduction of NFV in the telco domain. A tight relationship between the network service developer and the operator is no longer required. In this article, we provide two fundamental and necessary mechanisms that empower the developer in its new independent role: a modular MANO framework architecture that can be programmed per service or VNF by the developer and an SCK with software tools that ease the life of developers throughout a significantly enhanced NFV DevOps lifecycle.

To customise the proposed MANO framework, the developer can construct service or function specific managers. These managers are attached to the MANO when the associated network service or VNF is being instantiated, and they overwrite the generic MANO behaviour with customised workflows for lifecycle events of those specific network services and VNFs. The mechanism is secure and uses a public API. We included a use case where a content delivery network is extended with self adaptation capabilities through this mechanism. It shows that such programmability capabilities are essential for non-trivial network services to reach their full potential.

Secondly, we provided an SCK that enhances the NFV DevOps loop. The SCK contains tools that aid the service developer at every step: during development, when interfacing with operator infrastructure or when diagnosing test results. The Emulator, the core SCK component, emulates telco infrastructure on a local machine, enabling the developer to execute a portion of the tests locally without needing to interface with actual operator equipment. We show that our SCK meets all DevOps criteria during the entire development cycle, reduces development time as the introduction of the Emulator significantly shortens single development iterations, and out performs other available SCKs.

Acknowledgment

This work was funded through SONATA (671517) and 5GTANGO (761493), in the scope of the EC's Horizon 2020 and 5G-PPP programs. The expressed views are those of the authors only.

References

[1] "OSM Release Four Documentation," https://osm.etsi.org/wikipub/index.php/OSM_Release_FOUR_Documentation, (Accessed Oct. 10, 2018).

[2] G. Carella *et al.*, "Prototyping nfv-based multi-access edge computing in 5g ready networks with open baton," in *IEEE NetSoft*, 2017, pp. 1–4.

[3] ONAP, "Documentation of ONAP's CLAMP module," http://onap.readthedocs.io/en/latest/submodules/clamp.git/docs/index.html, (Accessed Oct. 10, 2018).

[4] S. Van Rossem *et al.*, "Introducing development features for virtualized network services," *IEEE Communications Magazine*, no. 99, pp. 2–10, 2018.

[5] R. Szabo *et al.*, "Elastic network functions: opportunities and challenges," *IEEE network*, vol. 29, no. 3, pp. 15–21, 2015.

[6] H. Karl *et al.*, "Devops for network function virtualisation: an architectural approach," *Trans. on Emerging Telecommunications Technologies*, vol. 27, no. 9, pp. 1206–1215, 2016.

[7] "Specific Manager API as defined in SONATA," https://github.com/sonata-nfv/son-mano-framework/wiki, (Accessed Oct. 10, 2018).

[8] S. Spinoso *et al.*, "Formal verification of virtual network function graphs in an sp-devops context," in *European Conference on Service-Oriented and Cloud Computing*, 2015, pp. 253–262.

[9] C. Meirosu *et al.*, "Devops for software-defined telecom infrastructures," *IETF*, pp. 1–20, 2015.

[10] A. Reddy, "Devops: The ibm approach," *Nova York: IBM Corporation*, 2013.

[11] W. John *et al.*, "Service provider devops," *IEEE Communications Magazine*, vol. 55, no. 1, pp. 204–211, 2017.

[12] S. Van Rossem *et al.*, "A network service development kit supporting the end-to-end lifecycle of nfv-based telecom services," in *IEEE NFV-SDN*, 2017, pp. 1–2.

[13] M. Peuster *et al.*, "Medicine: Rapid prototyping of production-ready network services in multi-pop environments," in *IEEE NFV-SDN*, 2016, pp. 148–153.

[14] I. Pelle *et al.*, "One tool to rule them all: A modular troubleshooting framework for sdn (and other) networks," in *ACM SIGCOMM*, 2015, p. 24.
[15] "Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; VNF Package specification," http://http://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/004/02.03.01_60/gs_nfv-sol004v020301p.pdf, (Accessed Oct. 10, 2018).

Thomas Soenen obtained his M.Sc. degree in Physics and Astronomy in 2012 from Ghent University. Currently, he is a researcher at IDLab, the Internet and Data science research group at Ghent University - imec. His interests focus on new network paradigms such as SDN and NFV.

Wouter Tavernier received his M.Sc. degree in Computer Science in 2002 from Ghent University, where he joined the IDLab group in 2006. He obtained a Ph.D. in 2012 and is currently professor at the same university. His interests focus on performance aspects of SDN, NFV and large-scale routing.

Manuel Peuster received his M.Sc. degree in computer science from Paderborn University in 2014, where he is a doctoral research associate in the Computer Networks group. His research interests are NFV, SDN, and performance benchmarking of distributed systems.

Felipe Vicens holds a Degree in Telecommunication Engineering from Andres Bello Catholic University (2009), where he was professor of the Telematics Lab (2009-2012). He is currently employed at Atos Research & Innovation. His interests are in SDN networks and cloud-native.

George Xilouris is a fellow researcher at Media Networks Lab, at the Institute of Informatics and Telecommunications at NCSR Demokritos. His interests are next generation networks and software networks.

Stavros Kolometsos is a research assistant at Media Networks Lab, at the Institute of Informatics and Telecommunications at NCSR Demokritos. Recent research activities include software networks and Service Function Chaining.

Michail-Alexandros Kourtis is a fellow researcher at Media Networks Lab, at the Institute of Informatics and Telecommunications at NCSR Demokritos. His research interests include NFV, SDN and Quality of Service.

Didier Colle is professor at Ghent University, where he received a Ph.D. in 2002 and a M.Sc. in electrotechnical engineering in 1997. His research interests are on fixed Internet architectures, optical networks and design of network algorithms.