Formal Verification of Virtual Network Function Graphs in an SP-DevOps Context

Serena Spinoso, Matteo Virgilio, Wolfgang John, Antonio Manzalini, Guido Marchetto, Riccardo Sisto

Formal verification of Virtual Network Function graphs in an SP-DevOps context

Serena Spinoso[1], Matteo Virgilio[1], Wolfgang John[2], Antonio Manzalini[3], Guido Marchetto[1], and Riccardo Sisto[1]

[1] DAUIN - Politecnico di Torino,
name.surname@polito.it
[2] Ericsson AB,
wolfgang.john@ericsson.com
[3] Strategy and Innovation - Future Centre,
antonio.manzalini@telecomitalia.com

Abstract. The role of software and its flexibility is becoming more and more important in todays networks. New emerging paradigms, such as Software Defined Networking (SDN) and Network Function Virtualization (NFV), are changing the rules of the game, shifting the focus on dynamicity and programmability. Perfectly aligned with this new spirit, the FP7 UNIFY European project aims at realizing this appealing vision by applying DevOps concepts to telecom operator networks and supporting the idea of fast network reconfiguration. However, the increased range of possibilities offered by the DevOps approach comes at the cost of designing new processes and toolkits to make SDN and NFV a concrete opportunity. In this paper we specifically focus on the verification process as part of the challenging tasks that must be addressed in this scenario and its fundamental role of automatically checking some desired network properties before deploying a particular configuration. Our preliminary results confirm the feasibility of the approach and encourage future efforts in this direction.

Keywords: DevOps, formal verification, service graphs, network function forwarding graph

1 Introduction

Ultra broadband diffusion, progresses in Information Technologies (IT), tumbling hardware costs and a wider and wider availability of open source software are shaping the evolution of Telecommunications and ICT infrastructures. In this context, paradigms such as Software Defined Network (SDN) and Network Function Virtualization (NFV) can be seen as expressions of a systemic trend called "Softwarization". Other expressions of the same trend are Cloud, Edge, and Fog Computing, Cloud Networking, etc. In essence, the disruptive innovation of "Softwarization" stands in the techno-economic feasibility of virtualizing most (if not all) network and service functions of Telecommunications and ICT

infrastructures. In this directions, it is argued that future Telecommunications infrastructures are likely to become highly dynamic, flexible and programmable production environments of ICT services. A first evaluation of this idea is carried out by the EU FP7 UNIFY[4] consortium, which sets out to integrate modern cloud computing and networking technologies by considering the entire network as a unified service production environment, spanning the vast networking assets and data centers of telecom providers. In order to reach a high level of agility for service innovation, UNIFY has one focus on providing dynamic service programming and orchestration, deploying logical service components, namely Virtual Network Functions (VNFs), across multiple network nodes. In particular, UNIFY architecture follows SDN principles with a logically centralized control and orchestration plane. Additionally, compute, storage and network abstractions are combined into a joint programmatic interface referred to as Network Function Forwarding Graph (NF-FG). An NF-FG defines a selected mapping of VNFs and their forwarding overlay definition into the virtualized resources presented by the underlying layer. Current OSS/BSS do not seem to cope with the requirements posed by this evolution: in fact, the operations of future Telecommunications infrastructures will involve the management and control of a myriad of software processes, rather than closed physical nodes. Thus, another important goal of UNIFY is the design and development of integrated operations and development capabilities under the name of Service Provider-DevOps (SP-DevOps). In fact, DevOps paradigm, formerly developed for Data Centers (DCs), is getting momentum as a source of inspiration regarding how to simplify and automate management processes for future Telecommunications infrastructures.

Among the above challenges, this paper focuses on the UNIFY verification process (i.e., the definition of methods and techniques to validate a particular network configuration before deploying it), which can be seen as an essential task in environments where reconfiguration of services is expected to be triggered very frequently, both in response to user requests and also in case of management events. Misconfiguration of dynamic network middleboxes[5], violation of specified network policies, or artificial insertion of malicious network functions are just examples of cases that a complete solution must properly handle in order to preserve network integrity and reliability. For this reason, the work presented in this paper goes in the direction of verifying complex graph of services through an intense modeling activity, targeted at the specific middleboxes and the network as a whole. We are motivated by the observation that most existing tools are "Openflow oriented", i.e. they mostly consider networks with a controller which installs <match, action> rules on the switches. Alternatively (and more generically but with the same fundamental limitations), they consider networks with devices that only perform forwarding decisions according to the packet header, i.e. without taking into account any additional traffic history information. Works as [6, 5, 9, 11] fall in this category and represent a valuable efforts in this research area. Our contribution is intended to move a step forward

[4] www.fp7-unify.eu

[5] In this paper we use the terms VNF and middlebox interchangeably.

and overcome the above mentioned limitations by extending these works. In this sense, one important reference is [8], which tackles exactly the same problem and provides a scalable solution based on an off-the-shelf SMT solver. We experiment with this approach and further develop it to meet our specific requirements, also enriching the available VNF models catalog in order to satisfy the demands for more and more complex service graphs and to validate the approach with different kinds of VNFs. We specifically consider the UNIFY use cases, but it is worth noticing how our work is much general and easily applicable to other scenarios since it involves very common network functions.

The rest of the paper is organized as follows. First, we introduce and clarify how and to which extent the DevOps approach can be applied in a network operator infrastructure (Section 2). After defining the processes needed to implement this vision, we move on our current approach to formally verify complex and rapid deployments of network function chains including a variety of middleboxes, deployed to augment the set of in-network services the operator is able to offer to its final customers (Section 3). In order to show our approach is feasible, we provide some preliminary performance evaluation results based on the extension of the above mentioned tool (Section 4). Section 5 finally concludes this work by summarizing our contribution and drawing up some possible near future directions.

2 The SP-DevOps concept

In order to cope with the high service velocity and increased dynamicity enabled by UNIFY and comparable SDN/NFV based environments, we consider a novel management and operation paradigm for Service Providers, called Service Provider DevOps - SP-DevOps. SP-DevOps is based on the same major underlying principles as identified for DevOps [10]: i) Monitor and validate operational quality; ii) Develop and test against production-like systems; iii) Deploy with repeatable, reliable processes; and iv) Amplify feedback loops. While we acknowledge that DevOps has also a crucial cultural dimension (reflected barely by the feedback loop principle), our work focuses on technical aspects associated to these principles, which reflect on processes and associated capabilities for integrated monitoring, verification, and testing software and programmable infrastructure. Even if significant parts of the telecommunication networks are foreseen to be virtualized in the future, we in [3] identified important characteristics of telecommunication networks that differ from traditional data centers, i.e.: (*i*) higher spatial distribution, as telecom resources are spread over wide areas due to coverage requirements; (*ii*) lower levels of redundancy in access and aggregation networks compared to the massive data centers of typical cloud computing companies; (*iii*) stronger requirements on high availability and latency in according to standards and customer expectations. These characteristics pose new challenges for applying DevOps principles in telecommunications environments [4]. SP-DevOps addresses them with a set of technical processes supporting developer and operator roles in a virtualized telecom network. Figure 1 illustrates the

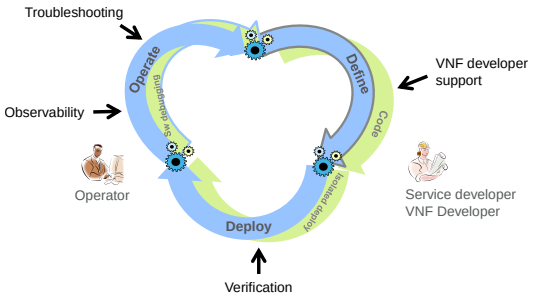

Fig. 1: SP-DevOps cycle for UNIFY service creation.

relation between the four SP-DevOps processes and the developer/operator roles by means of a service creation lifecycle. The four SP-DevOps processes follow the DevOps principles to meet specific challenges regarding Observability and Troubleshooting (Principle: Monitor and validate operation quality); Verification (Principle: Deploy with repeatable, reliable processes); and Development (Principle: Develop and test against production-like systems). We also identified three main roles involved in the processes: two Developer roles, where one is associated to a classical operator role assembling the service graph for a particular category of services (the Service Developer), and a second associated to the classical equipment vendor role in actually programming a VNF (the VNF Developer). The role of the Operator is to ensure that a set of performance indicators associated to a service are met when the service is deployed on virtual infrastructure within the domain of a telecom provider. SP-DevOps might not be a new form of DevOps as such, but it must include solutions that are uniquely tailored for the characteristics of its environment. Consequently, we propose the SP-DevOps Toolkit as an instantiation of the SP-DevOps concept [7]. The SP-DevOps Toolkit consists of a set of DevOps solutions that are developed targeting specific research challenges identified in the UNIFY production environment [4, 3]. Besides scalable and programmable infrastructure monitoring functions, the toolkit will also provide modules for deploy-time functional verification of various abstraction levels of service definition, supporting the three SP-DevOps roles. As in any development process, identification of problems early in the service or product livecycle can significantly reduce times and costs spent on complicated debugging and troubleshooting processes. In this paper, we focus on verification with respect to the service definitions and configurations initiated by the Service Developer. Automated verification functions operating during deploy-time on each layer of the orchestration and control architecture, facilitate verification as part of each step in the deployment process, allowing identification of problems early in the service lifecycle.

3 The verification process

The SP-DevOps paradigm represents a significant opportunity for service providers to implement more complex services in their networks and increase the agility by which a new function (or a chain of) can be automatically configured and deployed in their infrastructure. However, while the process of inserting and/or modifying functions throughout the network can be automated with technologies similar to the ones used for the Cloud Computing scenario [2], great importance has also to be placed on the design and implementation of automatic tools that can verify a network configuration on the fly, *before* it is deployed. For example, an operator may want to ensure that a given traffic flow is permitted (or not permitted, due to a policy constraint) from one node to another. Concerning this last aspect, our verification process is currently based on a verification approach recently proposed in [8]. In order to achieve high performance, this verification approach exploits Z3 [1], a state of the art SMT solver, and translates network scenarios with multiple middleboxes into sets of First Order Logic (FOL) formulas that are then analyzed by Z3. This choice is motivated by the overall verification tool performance and scalability, which would be hard to achieve with standard model checking based techniques. In fact, the latter requires time and memory that usually increase exponentially with the system complexity, while the SAT-based approach proposed in [8] seems to be less prone to this problem. The FOL formulas given to Z3 represent the network operating principles along with the functional behavior of all the VNFs involved in the scenario being considered. While [8] presents the general ideas of the proposed approach, not all the details are fully developed, and not all the different situations that may arise when considering different kinds of VNFs are considered. Here, we present our preliminary work towards integrating the approach presented in [8] into a SP-DevOps context like the one of UNIFY. A considerable part of this work has been about developing models for new VNFs that were not explicitly considered in [8], and making some first experiments with them.
In our design, the formal verification task is split into multiple sub-tasks, so that the whole process is simpler and faster. More precisely, at NF-FG deploy time, or when the graphs undergo modifications in response to higher level events (e.g., administration events or user requests), the VNF chains composing the graph are computed and then, for each of them, a formal model is generated, including the model of all the involved VNFs. Finally, the verification engine processes the whole VNF chain model to check the satisfiability of a given property. In particular, this paper focuses on reachability problems in service graphs, leaving the verification of other network properties as possible future work. Furthermore, since we are using abstract models of the real middleboxes, we assume that these models are correctly defined. This means that we verify abstract models of the real middleboxes, considering them as faithful representations of the real VNFs. Verification of possible mismatch between a VNF model and its implementation is out of scope for the current prototype. For further details about the adopted formal verification theory and other background concepts, please refer to [8].

$$(send(cache, n_0, p_0, t_0) \wedge \neg isInternal(n_0)) \implies \neg isInCache(p_0.url, t_0) \\ \wedge p_0.proto = HTTP\_REQ \wedge \exists(t_1, n_1) \mid (t_1 < t_0 \wedge isInternalNode(n_1) \\ \wedge recv(n_1, cache, p_0, t_1)), \forall n_0, p_0, t_0 \tag{1a}$$

$$(send(cache, n_0, p_0, t_0) \wedge isInternal(n_0)) \implies isInCache(p_0.url, t_0) \\ \wedge p_0.proto = HTTP\_RESP \wedge p_0.ip\_src = p_1.ip\_dest \wedge p_0.ip\_dest = p_1.ip\_src \wedge \\ \wedge \exists(p_1, t_1) \mid (t_1 < t_0 \wedge p_1.protocol = HTTP\_REQ \wedge p_1.url = p_0.url \\ \wedge recv(n_0, cache, p_1, t_1)), \forall n_0, p_0, t_0 \tag{1b}$$

$$isInCache(u_0, t_0) \implies \exists(t_1, t_2, p_1, p_2, n_1, n_1) \mid (t_1 < t_2 \wedge t_1 < t_0 \wedge t_2 < t_0 \\ \wedge recv(n_1, cache, p_1, t_1) \wedge recv(n_2, cache, p_2, t_2) \wedge p_1.proto = HTTP\_REQ \\ \wedge p_1.url = u_0 \wedge p_2.proto = HTTP\_RESP \wedge p_2.url = u_0 \wedge isInternal(n_2)) \\ \forall u_0, t_0 \tag{1c}$$

Fig. 2: Web cache model.

3.1 VNFs models

The approach for modeling network function chains proposed in [8] has been experimented by the authors of [8] with some middlebox types, such as stateless and stateful firewalls. When modelling scenarios that include VNFs that may alter packets (e.g. a NAT), it is necessary to also consider the possibility for a target VNF to receive a packet different from the one originally transmitted. This kind of situation regards a significant set of middleboxes that is currently deployed in SP networks and that is envisioned to be included in the NF-FG within the UNIFY project, e.g. NAT, VPN gateway and so on. We revisited the network constraints developed by the authors of [8], by introducing the possibility of verifying reachability properties between two network nodes and intermediate VNFs that do modify forwarded packet headers. Finally, we checked that verification works as expected with these revisited constraints, by experimenting with the new middlebox models that we developed.

The first VNF we consider is a simple web cache (reported in Figure 2). The functional model consists of two interfaces connected respectively to the private network, i.e., the one which contains the clients issuing HTTP requests, and the external network. Formula 1a states that a packet sent from the cache to a node belonging to the external network, implies a previous packet, containing a HTTP request and received from an internal node, which cannot be served by the cache (otherwise the request would have not been forwarded towards the external network). Formula 1b states that a packet sent from the cache to the internal network contains a `HTTP RESPONSE` for an URL which was in cache when the request has been received. We also state that the packet received from the internal network is a `HTTP REQUEST` and the target URL is the same as the response. The final formula expresses a constraint that the *isInCache()* function must respect. In particular, we state that a given URL (u_0) is in cache at time t_0 if (and only if) a request packet was received at time t_1 (where $t_1 < t_0$) for that URL and a subsequent packet was received at time t_2 (where $t_2 < t_0 \wedge t_2 > t_1$) carrying the corresponding `HTTP RESPONSE`.

$$
\begin{aligned}
&(send(nat, n_0, p_0, t_0) \wedge \neg isPrivateAddress(p_0.ip\_dest)) \implies p_0.ip\_src = ip\_nat \\
&\quad \wedge \exists (n_1, p_1, t_1) \mid (t_1 < t_0 \wedge recv(n_1, nat, p_1, t_1) \wedge isPrivateAddress(p_1.ip\_src) \\
&\quad \wedge p_1.origin = p_0.origin \wedge p_1.ip\_dest = p_0.ip\_dest \wedge p_1.seq\_no = p_0.seq\_no \\
&\quad \wedge p_1.proto = p_0.proto \wedge p_1.email\_from = p_0.email\_from \wedge p_1.url = p_0.url) \\
&\quad \forall n_0, p_0, t_0
\end{aligned} \tag{2a}
$$

$$
\begin{aligned}
&(send(nat, n_0, p_0, t_0) \wedge isPrivateAddress(p_0.ip_dest)) \implies \neg isPrivateAddress(p_0.ip\_src) \\
&\quad \wedge \exists (n_1, p_1, t_1) \mid (t_1 < t_0 \wedge recv(n_1, nat, p_1, t_1) \wedge \neg isPrivateAddress(p_1.ip\_src) \\
&\quad \wedge p_1.ip\_dest = ip\_nat \wedge p_1.ip\_src = p_0.ip\_src \wedge p_1.origin = p_0.origin \\
&\quad \wedge p_1.seq\_no = p_0.seq\_no \wedge p_1.proto = p_0.proto \wedge p_1.email\_from = p_0.email\_from \\
&\quad \wedge p_1.url = p_0.url) \wedge \exists (n_2, p_2, t_2) \mid (t_2 < t_1 \wedge recv(n_2, nat, p_2, t_2) \\
&\quad \wedge isPrivateAddress(p_2.ip\_src) \wedge p_2.ip\_dest = p_1.ip\_src \wedge p_2.ip\_dest = p_0.ip\_src \\
&\quad \wedge p_2.ip\_src = p_0.ip\_dest), \forall n_0, p_0, t_0
\end{aligned} \tag{2b}
$$

Fig. 3: NAT model.

The second middlebox we modeled is the NAT function. The corresponding model is reported in Figure 3. In order to model the NAT behaviour, a distinction between the private and external network is needed. This separation is modeled by using a boolean function (*isPrivateAddress()*) that returns true if a given IP address belongs to the set of internal node addresses. Analyzing the reported formulas, we start by considering an internal node which initiates a communication with an external node (Formula 2a). In this case, the NAT sends a packet (p_0) to an external IP address, if and only if it has previously received a packet (p_1) from an internal node. The received and sent packets must be equal for all fields, except for the *ip_src*, which must be equal to the NAT public IP address.
On the other hand, the traffic in the opposite direction (from the external network to the private) is modeled by the Formula 2b. In this case, we state that if the NAT is sending a packet to an internal address, this packet (p_0) must have an external IP address as its source. Moreover, p_0 must be preceded by another packet (p_1 in the formula), which is, in turn, received by the NAT and it is equal to p_0 for all the other fields. It is worth noting that, generally, a communication between internal and external nodes cannot be started by the external node in presence of a NAT. As a consequence, this condition is expressed in the Formula 2b by imposing that p_1 must be preceded by another packet p_2, sent to the NAT from an internal node.

4 Preliminary results

In order to evaluate the new developed models and the overall approach, we consider the NF-FG[6] shown in Figure 4 as a use case. In our reference graph, four end-hosts (two clients and two servers) can generate either HTTP or POP3 and

[6] We do not provide the firewall VNF model as it was presented as use case in [8].

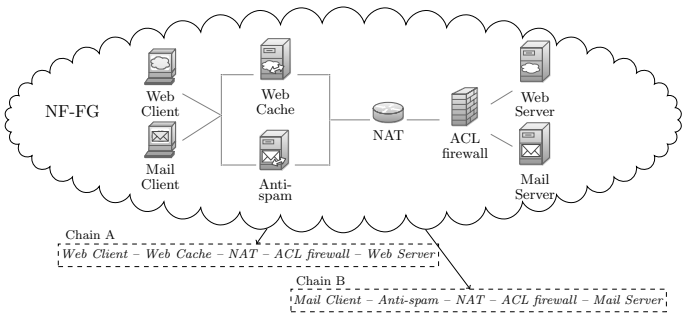

Fig. 4: An example of Network Function-Forwarding Graph.

also SMTP traffic, which is processed by different middleboxes when traversing the graph. Moreover, some of those network functions may require a different configuration. Specifically, the NAT must be configured in order to know which hosts belong to the private network (as the web cache) and which IP address must be used as masquerading address; the firewall must be provided with a set of ACL entries that specify which couples of nodes are authorized to exchange traffic. Additionally, the forwarding is configured such that the web traffic is forwarded to the web cache, while the email traffic (both POP3 and SMTP) is routed to an anti-spam function. A first step towards the NF-FG verification is the VNF chains extraction. In our use case, two chains are extracted from the NF-FG (Figure 4): the *Chain A* processes the web traffic, while the *Chain B* is traversed by POP3 and SMTP packets.

We perform multiple tests on the two chains to cover different cases and configuration options: *(i)* anti-spam and firewall configurations and *(ii)* traffic directions (from client to server and vice-versa). Concerning the Chain A, only the ACL firewall can be configured, hence we setup two tests: one with the firewall configured to allow all the traffic (test A.1) and the other one with the firewall configured to drop all packets exchanged between the web client and server (test A.2).

Instead the Chain B is tested in three scenarios, obtained by changing the firewall and anti-spam configurations as follows: (*i*) test B.1, similarly to test A.1, is performed without any function configured to drop the received traffic; (*ii*) in test B.2, the firewall drops the traffic between the mail client and server (Figure 5); (*iii*) test B.3 is such that the anti-spam is configured to drop all the emails sent by the mail client, while the traffic originated by the server is allowed (Figure 5). Our evaluation is executed on a workstation with 32GB of RAM and an Intel i7-3770 CPU running an Ubuntu 14.04.01 with kernel 3.13.0-24-generic. The results are shown in Figure 5, where the verification time is reported for each presented scenario.

In test A.1 the reachability problem from the client to the server (the light grey colored bar in Figure 5) is satisfied as expected. It is worth noting that the unsatisfiability of the problem in the opposite direction (the dark grey colored bar

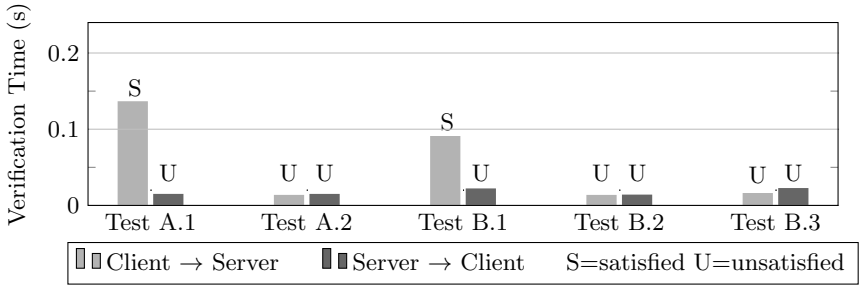

Fig. 5: **Test {A, B}.1**: firewall and anti-spam configured to accept packets; **Test {A, B}.2**: firewall configured to drop server/client packets; **Test B.3**: anti-spam configured to drop server/client packets.

in Figure 5) is due to the fact that client and server can exchange traffic only if the connection is initiated by the client. In test A.2, in both cases the reachability problems are not satisfied because of the firewall VNF configuration. In test B.1, the verification problem is satisfiable in case of traffic sent by the mail client, while the reachability property is not verified for the traffic sent by the mail server for the above-mentioned reasons.
As it can be seen from the achieved results, performance is promising also in the worst case scenario, since we are able to solve the reachability problem in less than 200ms, while the verification time is less than 50ms in most cases. This is reasonably in line with the UNIFY requirements, especially in terms of time required by the verification process to authorize a newly asked network reconfiguration.

5 Conclusion

It is argued that in the future Telecommunications infrastructures are likely to become highly dynamic, flexible and programmable production environments capable of providing any ICT services. Future operations will involve the management and control of a myriad of software processes, rather than closed physical nodes.

In fact, today most SPs still have rather complicated and static operational processes. DevOps, formerly developed for managing Data Centers (DCs), is attracting a growing interests as a paradigm to be extended to future Telecommunications infrastructures. Nevertheless, it is argued that the DevOps will jump ahead current ossification only if it will be sustainable from a business viewpoint (CAPEX, OPEX saving are not enough): importantly DevOps criteria of success depend on how closely the related future infrastructures (e.g. UNIFY) will be capable of enabling new service paradigms for SP's (e.g., Immersive Communications, Anything as a Service, etc). Motivated by these considerations, in this paper we presented our initial contribution related to the verification process on

service graphs, which is one of the most important pillars in the SP-DevOps feedback cycle. After generalizing the applicability of a state of the art approach to the verification of complex network graph, we presented and discussed a couple of models we developed to validate our key ideas. Given the promising evaluation results achieved, we plan to address more efforts to some open topics in the middlebox verification area such as scalability issues in verifying complex service graphs and significant opportunities to optimize the verification process when incremental service graph modifications come into play.

6 Acknowledgment

This work was conducted within the framework of the FP7 UNIFY project, which is partially funded by the Commission of the European Union. Study sponsors had no role in writing this report. The views expressed do not necessarily represent the views of the authors' employers, the UNIFY project, or the Commission of the European Union.

References

1. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. pp. 337–340. TACAS'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg (2008)
2. Jain, R., Paul, S.: Network virtualization and software defined networking for cloud computing: a survey. Communications Magazine, IEEE 51(11) (November 2013)
3. John, W., Meirosu, C.: Unify d4.1: Initial requirements for the sp-devops concept, universal node capabilities and proposed tools (2014), `https://www.fp7-unify.eu/index.php/results.html#Deliverables`
4. John, W., Pentikousis, K., Agapiou, G., Jacob, E., Kind, M., Manzalini, A., Risso, F., Staessens, D., Steinert, R., Meirosu, C.: Research directions in network service chaining. In: SDN4FNS, 2013 IEEE SDN for (Nov 2013)
5. Kazemian, P., Varghese, G., McKeown, N.: Header space analysis: Static checking for networks. In: NSDI 12. USENIX, San Jose, CA (2012)
6. Khurshid, A., Zou, X., Zhou, W., Caesar, M., Godfrey, P.B.: Veriflow: Verifying network-wide invariants in real time. In: NSDI 13. USENIX, Lombard, IL (2013)
7. Meirosu, C.: m4.1: Sp-devops concept evolution and initial plans for prototyping (2014), `https://www.fp7-unify.eu/index.php/results.html#Deliverables`
8. Panda, A., Lahav, O., Argyraki, K.J., Sagiv, M., Shenker, S.: Verifying isolation properties in the presence of middleboxes. CoRR abs/1409.7687 (2014)
9. Porras, P., Shin, S., Yegneswaran, V., Fong, M., Tyson, M., Gu, G.: A security enforcement kernel for openflow networks. HotSDN '12, ACM, New York, NY, USA (2012)
10. Sharma, S., Coyne, B.: DevOps For Dummies. Limited IBM Edition' book (October 2013)
11. Son, S., Shin, S., Yegneswaran, V., Porras, P.A., Gu, G.: Model checking invariant security properties in openflow. In: ICC. pp. 1974–1979. IEEE (2013)