NFVRG
Internet Draft
Intended status: Informational
Expires: September 2016

C. Meirosu
Ericsson
A. Manzalini
Telecom Italia
R. Steinert
SICS
G. Marchetto
Politecnico di Torino
I. Papafili
Hellenic Telecommunications Organization
K. Pentikousis
EICT
S. Wright
AT&T

March 20, 2016March 18, 2016

DevOps for Software-Defined Telecom Infrastructures
draft-unify-nfvrg-devops-04.txt

Status of this Memo

Abstract

Carrier-grade network management was optimized for environments built with monolithic physical nodes and involves significant deployment, integration and maintenance efforts from network service providers. The introduction of virtualization technologies, from the physical layer all the way up to the application layer, however, invalidates several well-established assumptions in this domain. This draft opens the discussion in NFVRG about challenges related to transforming the telecom network infrastructure into an agile, model-driven production environment for communication services. We take inspiration from data center DevOps regarding how to simplify and automate management processes for a telecom service provider software-defined infrastructure (SDI). Among the identified challenges, we consider scalability of observability processes and automated inference of monitoring requirements from logical forwarding graphs, as well as initial placement (and re-placement) of monitoring functionality following changes in flow paths enforced by the controllers. In another category of challenges, verifying correctness of behavior for network functions where flow rules are no longer necessary and sufficient for determining the forwarding state (for example, stateful firewalls or load balancers) is very difficult with current technology. Finally, we introduce challenges associated with operationalizing DevOps principles at scale in software-defined telecom networks in three areas related to key monitoring, verification and troubleshooting processes.

Table of Contents

1. Introduction

Carrier-grade network management was developed as an incremental solution once a particular network technology matured and came to be deployed in parallel with legacy technologies. This approach requires significant integration efforts when new network services are launched. Both centralized and distributed algorithms have been developed in order to solve very specific problems related to configuration, performance and fault management. However, such algorithms consider a network that is by and large functionally static. Thus, management processes related to introducing new or maintaining functionality are complex and costly due to significant efforts required for verification and integration.

Network virtualization, by means of Software-Defined Networking (SDN) and Network Function Virtualization (NFV), creates an environment where network functions are no longer static or strictly embedded in physical boxes deployed at fixed points. The virtualized network is dynamic and open to fast-paced innovation enabling efficient network management and reduction of operating cost for network operators. A significant part of network capabilities are expected to become available through interfaces that resemble the APIs widespread within

datacenters instead of the traditional telecom means of management such as the Simple Network Management Protocol, Command Line Interfaces or CORBA. Such an API-based approach, combined with the programmability offered by SDN interfaces [RFC7426], open opportunities for handling infrastructure, resources, and Virtual Network Functions (VNFs) as code, employing techniques from software engineering.

The efficiency and integration of existing management techniques in virtualized and dynamic network environments are limited, however. Monitoring tools, e.g. based on simple counters, physical network taps and active probing, do not scale well and provide only a small part of the observability features required in such a dynamic environment. Although huge amounts of monitoring data can be collected from the nodes, the typical granularity is rather coarse. Debugging and troubleshooting techniques developed for software-defined environments are a research topic that has gathered interest in the research community in the last years. Still, it is yet to be explored how to integrate them into an operational network management system. Moreover, research tools developed in academia (such as NetSight [H2014], OFRewind [W2011], FlowChecker [S2010], etc.) were limited to solving very particular, well-defined problems, and oftentimes are not built for automation and integration into carrier-grade network operations workflows.

The topics at hand have already attracted several standardization organizations to look into the issues arising in this new environment. For example, IETF working groups have activities in the area of OAM and Verification for Service Function Chaining [I-D.aldrin-sfc-oam-framework] [I-D.lee-sfc-verification] for Service Function Chaining. At IRTF, [RFC7149] asks a set of relevant questions regarding operations of SDNs. The ETSI NFV ISG defines the MANO interfaces [NFVMANO], and TMForum investigates gaps between these interfaces and existing specifications in [TR228]. The need for programmatic APIs in the orchestration of compute, network and storage resources is discussed in [I-D.unify-nfvrg-challenges].

From a research perspective, problems related to operations of software-defined networks are in part outlined in [SDNsurvey] and research referring to both cloud and software-defined networks are discussed in [D4.1].

The purpose of this first version of this document is to act as a discussion opener in NFVRG by describing a set of principles that are relevant for applying DevOps ideas to managing software-defined telecom network infrastructures. We identify a set of challenges related to developing tools, interfaces and protocols that would

support these principles and how can we leverage standard APIs for simplifying management tasks.

2. Software-Defined Telecom Infrastructure: Roles and DevOps principles

Agile methods used in many software focused companies are focused on releasing small interactions of code to implement VNFs with high velocity and high quality into a production environment. Similarly, Service providers are interested to release incremental improvements in the network services that they create from virtualized network functions. The cycle time for devops as applied in many open source projects is on the order of one quarter year or 13 weeks.

The code needs to undergo a significant amount of automated testing and verification with pre-defined templates in a realistic setting. From the point of view of infrastructure management, the verification of the network configuration as result of network policy decomposition and refinement, as well as the configuration of virtual functions, is one of the most sensitive operations. When troubleshooting the cause of unexpected behavior, fine-grained visibility onto all resources supporting the virtual functions (either compute, or network-related) is paramount to facilitating fast resolution times. While compute resources are typically very well covered by debugging and profiling toolsets based on many years of advances in software engineering, programmable network resources are a still a novelty and tools exploiting their potential are scarce.

2.1. Service Developer Role

We identify two dimensions of the "developer" role in software-defined infrastructure (SDI). One dimension relates to determining which high-level functions should be part of a particular service, deciding what logical interconnections are needed between these blocks and defining a set of high-level constraints or goals related to parameters that define, for instance, a Service Function Chain. This could be determined by the product owner for a particular family of services offered by a telecom provider. Or, it might be a key account representative that adapts an existing service template to the requirements of a particular customer by adding or removing a small number of functional entities. We refer to this person as the Service Developer and for simplicity (access control, training on technical background, etc.) we consider the role to be internal to the telecom provider.

2.2. VNF Developer role

Another dimension of the "developer" role is a person that writes the software code for a new virtual network function (VNF). Depending on the actual VNF being developed, this person might be internal or external (e.g. a traditional equipment vendor) to the telecom provider. We refer to them as VNF Developers.

2.3. System Integrator role

The System Integrator role is to some extent similar to the Service Developer: people in this role need to identify the components of the system to be delivered. However, for the Service Developer, the service components are pre-integrated meaning that they have the right interfaces to interact with each other. In contrast, the Systems Integrator needs to develop the software that makes the system components interact with each other. As such, the Systems Integrator role combines aspects of the Developer roles and adds yet another dimension to it. Compared to the other Developer roles, the System Integrator might face additional challenges due to the fact that they might not have access to the source code of some of the components. This limits for example how fast they could address issues with components to be integrated, as well as uneven workload depending on the release granularity of the different components that need to be integrated.

2.4. Operator role

The role of an Operator in SDI is to ensure that the deployment processes were successful and a set of performance indicators associated to a service are met while the service is supported on virtual infrastructure within the domain of a telecom provider.

2.5. Customer role

A Customer contracts a telecom operator to provide one or more services. In SDI, the Customer may communicate with the provider through an online portal. Compared to the Service Developer, the Customer is external to the operator and may define changes to their own service instance only in accordance to policies defined by the Service Developer. In addition to the usual per-service utilization statistics, in SDI the portal may enable the customer to trigger certain performance management or troubleshooting tools for the service. This, for example, enables the Customer to determine whether the root cause of certain error or degradation condition that they observe is located in the telecom operator domain or not and may facilitate the interaction with the customer support teams.

2.6. DevOps Principles

In line with the generic DevOps concept outlined in [DevOpsP], we consider that these four principles as important for adapting DevOps ideas to SDI:

* Deploy with repeatable, reliable processes: Service and VNF Developers should be supported by automated build, orchestrate and deploy processes that are identical in the development, test and production environments. Such processes need to be made reliable and trusted in the sense that they should reduce the chance of human error and provide visibility at each stage of the process, as well as have the possibility to enable manual interactions in certain key stages.

* Develop and test against production-like systems: both Service Developers and VNF Developers need to have the opportunity to verify and debug their respective SDI code in systems that have characteristics which are very close to the production environment where the code is expected to be ultimately deployed. Customizations of Service Function Chains or VNFs could thus be released frequently to a production environment in compliance with policies set by the Operators. Adequate isolation and protection of the services active in the infrastructure from services being tested or debugged should be provided by the production environment.

* Monitor and validate operational quality: Service Developers, VNF Developers and Operators must be equipped with tools, automated as much as possible, that enable to continuously monitor the operational quality of the services deployed on SDI. Monitoring tools should be complemented by tools that allow verifying and validating the operational quality of the service in line with established procedures which might be standardized (for example, Y.1564 Ethernet Activation [Y1564]) or defined through best practices specific to a particular telecom operator.

* Amplify development cycle feedback loops: An integral part of the DevOps ethos is building a cross-cultural environment that bridges the cultural gap between the desire for continuous change by the Developers and the demand by the Operators for stability and reliability of the infrastructure. Feedback from customers is collected and transmitted throughout the organization. From a technical perspective, such cultural aspects could be addressed through common sets of tools and APIs that are aimed at providing a shared vocabulary for both Developers and Operators, as well as simplifying the reproduction of problematic situations in the development, test and operations environments.

Network operators that would like to move to agile methods to deploy and manage their networks and services face a different environment compared to typical software companies where simplified trust relationships between personnel are the norm. In software companies, it is not uncommon that the same person may be rotating between different roles. In contrast, in a telecom service provider, there are strong organizational boundaries between suppliers (whether in Developer roles for network functions, or in Operator roles for outsourced services) and the carrier's own personnel that might also take both Developer and Operator roles. How DevOps principles reflect on these trust relationships and to what extent initiatives such as co-creation could transform the environment to facilitate closer Dev and Ops integration across business boundaries is an interesting area for business studies, but we could not for now identify a specific technological challenge.

3. Continuous Integration

Software integration is the process of bringing together the software component subsystems into one software system, and ensuring that the subsystems function together as a system. Software integration can apply regardless of the size of the software components. The objective of Continuous Integration is to prevent integration problems close to the expected release of a software development project into a production (operations) environment. Continuous Integration is therefore closely coupled with the notion of DevOps as a mechanism to ease the transition from development to operations.

Continuous integration may result in multiple builds per day. It is also typically used in conjunction with test driven development approaches that integrate unit testing into the build process. The unit testing is typically automated through build servers. Such servers may implement a variety of additional static and dynamic tests as well as other quality control and documentation extraction functions. The reduced cycle times of continuous enable improved software quality by applying small efforts frequently.

Continuous Integration applies to developers of VNF as they integrate the components that they need to deliver their VNF. The VNFs may contain components developed by different teams within the VNF Provider, or may integrate code developed externally - e.g. in commercial code libraries or in open source communities.

Service providers also apply continuous integration in the development of network services. Network services are comprised of

various aspects including VNFs and connectivity within and between them as well as with various associated resource authorizations. The components of the networks service are all dynamic, and largely represented by software that must be integrated regularly to maintain consistency. Some of the software components that Service Providers may be sourced from VNF Providers or from open source communities. Service Providers are increasingly motivated to engage with open Source communities [OSandS]. Open source interfaces supported by open source communities may be more useful than traditional paper interface specifications. Even where Service Providers are deeply engaged in the open source community (e.g. OPNFV) many service providers may prefer to obtain the code through some software provider as a business practice. Such software providers have the same interests in software integration as other VNF providers.

4. Continuous Delivery

The practice of Continuous Delivery extends Continuous Integration by ensuring that the software (either a VNF code or code for SDI) checked in on the mainline is always in a user deployable state and enables rapid deployment by those users. For critical systems such as telecommunications networks, Continuous Delivery has the advantage of including a manual trigger before the actual deployment in the live system, compared to the Continuous Deployment methodology which is also part of DevOps processes in software companies.

5. Consistency, Availability and Partitioning Challenges

The CAP theorem [CAP] states that any networked shared-data system can have at most two of following three properties: 1) Consistency (C) equivalent to having a single up-to-date copy of the data; 2) high Availability (A) of that data (for updates); and 3) tolerance to network Partitions (P).

Looking at a telecom SDI as a distributed computational system (routing/forwarding packets can be seen as a computational problem), just two of the three CAP properties will be possible at the same time. The general idea is that 2 of the 3 have to be chosen. CP favor consistency, AP favor availability, CA there are no partition. This has profound implications for technologies that need to be developed in line with the "deploy with repeatable, reliable processes" principle for configuring SDI states. Latency or delay and partitioning properties are closely related, and such relation becomes more important in the case of telecom service providers where Devs and Ops interact with widely distributed infrastructure.

Limitations of interactions between centralized management and distributed control need to be carefully examined in such environments. Traditionally connectivity was the main concern: C and A was about delivering packets to destination. The features and capabilities of SDN and NFV are changing the concerns: for example in SDN, control plane Partitions no longer imply data plane Partitions, so A does not imply C. In practice, CAP reflects the need for a balance between local/distributed operations and remote/centralized operations.

Furthermore to CAP aspects related to individual protocols, interdependencies between CAP choices for both resources and VNFs that are interconnected in a forwarding graph need to be considered. This is particularly relevant for the "Monitor and Validate Operational Quality" principle, as apart from transport protocols, most OAM functionality is generally configured in processes that are separated from the configuration of the monitored entities. Also, partitioning in a monitoring plane implemented through VNFs executed on compute resources does not necessarily mean that the dataplane of the monitored VNF was partitioned as well.

6. Stability Challenges

The dimensions, dynamicity and heterogeneity of networks are growing continuously. Monitoring and managing the network behavior in order to meet technical and business objectives is becoming increasingly complicated and challenging, especially when considering the need of predicting and taming potential instabilities.

In general, instability in networks may have primary effects both jeopardizing the performance and compromising an optimized use of resources, even across multiple layers: in fact, instability of end-to-end communication paths may depend both on the underlying transport network, as well as the higher level components specific to flow control and dynamic routing. For example, arguments for introducing advanced flow admission control are essentially derived from the observation that the network otherwise behaves in an inefficient and potentially unstable manner. Even with resources over provisioning, a network without an efficient flow admission control has instability regions that can even lead to congestion collapse in certain configurations. Another example is the instability which is characteristic of any dynamically adaptive routing system. Routing instability, which can be (informally) defined as the quick change of network reachability and topology information, has a number of possible origins, including problems with connections, router

failures, high levels of congestion, software configuration errors, transient physical and data link problems, and software bugs.

As a matter of fact, the states monitored and used to implement the different control and management functions in network nodes are governed by several low-level configuration commands (today still done mostly manually). Further, there are several dependencies among these states and the logic updating the states (most of which are not kept aligned automatically). Normally, high-level network goals (such as the connectivity matrix, load-balancing, traffic engineering goals, survivability requirements, etc) are translated into low-level configuration commands (mostly manually) individually executed on the network elements (e.g., forwarding table, packet filters, link-scheduling weights, and queue-management parameters, as well as tunnels and NAT mappings). Network instabilities due to configuration errors can spread from node to node and propagate throughout the network.

DevOps in the data center is a source of inspiration regarding how to simplify and automate management processes for software-defined infrastructure. Although the low-level configuration could be automated by DevOps tools such as CFEngine [C2015], Puppet [P2015] and Ansible [A2015], the high-level goal translation towards tool-specific syntax is still a manual process. In addition, while carrier-grade configuration tools using the NETCONF protocol support complex atomic transaction management (which reduces the potential for instability), Ansible requires third-party components to support rollbacks and the Puppet transactions are not atomic.

As a specific example, automated configuration functions are expected to take the form of a "control loop" that monitors (i.e., measures) current states of the network, performs a computation, and then reconfigures the network. These types of functions must work correctly even in the presence of failures, variable delays in communicating with a distributed set of devices, and frequent changes in network conditions. Nevertheless cascading and nesting of automated configuration processes can lead to the emergence of non-linear network behaviors, and as such sudden instabilities (i.e. identical local dynamic can give rise to widely different global dynamics).

7. Observability Challenges

Monitoring algorithms need to operate in a scalable manner while providing the specified level of observability in the network, either for operation purposes (Ops part) or for debugging in a development phase (Dev part). We consider the following challenges:

* Scalability - relates to the granularity of network observability, computational efficiency, communication overhead, and strategic placement of monitoring functions.

* Distributed operation and information exchange between monitoring functions - monitoring functions supported by the nodes may perform specific operations (such as aggregation or filtering) locally on the collected data or within a defined data neighborhood and forward only the result to a management system. Such operation may require modifications of existing standards and development of protocols for efficient information exchange and messaging between monitoring functions. Different levels of granularity may need to be offered for the data exchanged through the interfaces, depending on the Dev or Ops role. Modern messaging systems, such as Apache Kafka [AK2015], widely employed in datacenter environments, were optimized for messages that are considerably larger than reading a single counter value (typical SNMP GET call usage) - note the throughput vs record size from [K2014]. It is also debatable to what extent properties such as message persistence within the bus are needed in a carrier environment, where MIBs practically offer already a certain level of persistence of management data at the node level. Also, they require the use of IP addressing which might not be needed when the monitored data is consumed by a function within the same node.

* Common communication channel between monitoring functions and higher layer entities (orchestration, control or management systems) - a single communication channel for configuration and measurement data of diverse monitoring functions running on heterogeneous hard- and software environments. In telecommunication environments, infrastructure assets span not only large geographical areas, but also a wide range of technology domains, ranging from CPEs, access-, aggregation-, and transport networks, to datacenters. This heterogeneity of hard- and software platforms requires higher layer entities to utilize various parallel communication channels for either configuration or data retrieval of monitoring functions within these technology domains. To address automation and advances in monitoring programmability, software defined telecommunication infrastructures would benefit from a single flexible communication channel, thereby supporting the dynamicity of virtualized environments. Such a channel should ideally support propagation of

configuration, signalling, and results from monitoring functions; carrier-grade operations in terms of availability and multi-tenant features; support highly distributed and hierarchical architectures, keeping messages as local as possible; be lightweight, topology independent, network address agnostic; support flexibility in terms of transport mechanisms and programming language support.
Existing popular state-of-the-art message queuing systems such as RabbitMQ [R2015] fulfill many of these requirements. However, they utilize centralized brokers, posing a single point-of-failure and scalability concerns within vastly distributed NFV environment. Furthermore, transport support is limited to TCP/IP. ZeroMQ [Z2015] on the other hard lacks any advanced features for carrier-grade operations, including high-availability, authentication, and tenant isolation.

* Configurability and conditional observability - monitoring functions that go beyond measuring simple metrics (such as delay, or packet loss) require expressive monitoring annotation languages for describing the functionality such that it can be programmed by a controller. Monitoring algorithms implementing self-adaptive monitoring behavior relative to local network situations may employ such annotation languages to receive high-level objectives (KPIs controlling tradeoffs between accuracy and measurement frequency, for example) and conditions for varying the measurement intensity. Steps in this direction were taken by the DevOps tools such as Splunk [S2015], whose collecting agent has the ability to load particular apps that in turn access specific counters or log files. However, such apps are tool specific and may also require deploying additional agents that are specific to the application, library or infrastructure node being monitored. Choosing which objects to monitor in such environment means deploying a tool-specific script that configures the monitoring app.

* Automation - includes mapping of monitoring functionality from a logical forwarding graph to virtual or physical instances executing in the infrastructure, as well as placement and re-placement of monitoring functionality for required observability coverage and configuration consistency upon updates in a dynamic network environment. Puppet [P2015] manifests or Ansible [A2015] playbooks could be used for automating the deployment of monitoring agents, for example those used by Splunk [S2015]. However, both manifests and playbooks were designed to represent the desired system configuration snapshot at a particular moment in time - they would now need to be generated automatically by the orchestration tools instead of a DevOps person.

* Actionable data

Data produced by observability tools could be utilized in a wide category of processes, ranging from billing and dimensioning to real-time troubleshooting and optimization. In order to allow for data-driven automated decisions and actuations based on these decisions, the data needs to be actionable. We define actionable data as being representative for a particular context or situation and an adequate input towards a decision. Ensuring actionable data is challenging in a number of ways, including: defining adaptive correlation and sampling windows, filtering and aggregation methods that are adapted or coordinated with the actual consumer of the data, and developing analytical and predictive methods that account for the uncertainty or incompleteness of the data.

* Data Virtualization

Data is key in helping both Developers and Operators perform their tasks. Traditional Network Management Systems were optimized for using one database that contains the master copy of the operational statistics and logs of network nodes. Ensuring access to this data from across the organization is challenging because strict privacy and business secrets need to be protected. In DevOps-driven environments, data needs to be made available to Developers and their test environments. Data virtualization collectively defines a set of technologies that ensure that restricted copies of the partial data needed for a particular task may be made available while enforcing strict access control. Further than simple access control, data virtualization needs to address scalability challenges involved in copying large amounts of operational data as well as automatically disposing of it when the task authorized for using it has finished.

8. Verification Challenges

Enabling ongoing verification of code is an important goal of continuous integration as part of the data center DevOps concept. In a telecom SDI, service definitions, decompositions and configurations need to be expressed in machine-readable encodings. For example, configuration parameters could be expressed in terms of YANG data models. However, the infrastructure management layers (such as Software-Defined Network Controllers and Orchestration functions) might not always export such machine-readable descriptions of the runtime configuration state. In this case, the management layer itself could be expected to include a verification process that has the same challenges as the stand-alone verification processes we outline later in this section. In that sense, verification can be considered as a set of features providing gatekeeper functions to

verify both the abstract service models and the proposed resource configuration before or right after the actual instantiation on the infrastructure layer takes place.

A verification process can involve different layers of the network and service architecture. Starting from a high-level verification of the customer input (for example, a Service Graph as defined in [I-D.unify-nfvrg-challenges]), the verification process could go more in depth to reflect on the Service Function Chain configuration. At the lowest layer, the verification would handle the actual set of forwarding rules and other configuration parameters associated to a Service Function Chain instance. This enables the verification of more quantitative properties (e.g. compliance with resource availability), as well as a more detailed and precise verification of the abovementioned topological ones. Existing SDN verification tools could be deployed in this context, but the majority of them only operate on flow space rules commonly expressed using OpenFlow syntax.

Moreover, such verification tools were designed for networks where the flow rules are necessary and sufficient to determine the forwarding state. This assumption is valid in networks composed only by network functions that forward traffic by analyzing only the packet headers (e.g. simple routers, stateless firewalls, etc.). Unfortunately, most of the real networks contain active network functions, represented by middle-boxes that dynamically change the forwarding path of a flow according to function-local algorithms and an internal state (that is based on the received packets), e.g. load balancers, packet marking modules and intrusion detection systems. The existing verification tools do not consider active network functions because they do not account for the dynamic transformation of an internal state into the verification process.

Defining a set of verification tools that can account for active network functions is a significant challenge. In order to perform verification based on formal properties of the system, the internal states of an active (virtual or not) network function would need to be represented. Although these states would increase the verification process complexity (e.g., using simple model checking would not be feasible due to state explosion), they help to better represent the forwarding behavior in real networks. A way to address this challenge is by attempting to summarize the internal state of an active network function in a way that allows for the verification process to finish within a reasonable time interval.

9. Troubleshooting Challenges

One of the problems brought up by the complexity introduced by NFV and SDN is pinpointing the cause of a failure in an infrastructure that is under continuous change. Developing an agile and low-maintenance debugging mechanism for an architecture that is comprised of multiple layers and discrete components is a particularly challenging task to carry out. Verification, observability, and probe-based tools are key to troubleshooting processes, regardless whether they are followed by Dev or Ops personnel.

* Automated troubleshooting workflows

Failure is a frequently occurring event in network operation. Therefore, it is crucial to monitor components of the system periodically. Moreover, the troubleshooting system should search for the cause automatically in the case of failure. If the system follows a multi-layered architecture, monitoring and debugging actions should be performed on components from the topmost layer to the bottom layer in a chain. Likewise, the result of operations should be notified in reverse order. In this regard, one should be able to define monitoring and debugging actions through a common interface that employs layer hopping logic. Besides, this interface should allow fine-grained and automatic on-demand control for the integration of other monitoring and verification mechanisms and tools.

* Troubleshooting with active measurement methods

Besides detecting network changes based on passively collected information, active probes to quantify delay, network utilization and loss rate are important to debug errors and to evaluate the performance of network elements. While tools that are effective in determining such conditions for particular technologies were specified by IETF and other standardization organization, their use requires a significant amount of manual labor in terms of both configuration and interpretation of the results.

In contrast, methods that test and debug networks systematically based on models generated from the router configuration, router interface tables or forwarding tables, would significantly simplify management. They could be made usable by Dev personnel that have little expertise on diagnosing network defects. Such tools naturally lend themselves to integration into complex troubleshooting workflows that could be generated automatically based on the description of a particular service chain. However, there are scalability challenges associated with deploying such tools in a network. Some tools may poll each networking device for the forwarding table information to

calculate the minimum number of test packets to be transmitted in the network. Therefore, as the network size and the forwarding table size increase, forwarding table updates for the tools may put a non-negligible load in the network.

10. Programmable network management

The ability to automate a set of actions to be performed on the infrastructure, be it virtual or physical, is key to productivity increases following the application of DevOps principles. Previous sections in this document touched on different dimensions of programmability:

- Section 5 approached programmability in the context of developing new capabilities for monitoring and for dynamically setting configuration parameters of deployed monitoring functions

- Section 7 reflected on the need to determine the correctness of actions that are to be inflicted on the infrastructure as result of executing a set of high-level instructions

- Section 8 considered programmability in the perspective of an interface to facilitate dynamic orchestration of troubleshooting steps towards building workflows and for reducing the manual steps required in troubleshooting processes

We expect that programmable network management - along the lines of [RFC7426] - will draw more interest as we move forward. For example, in [I-D.unify-nfvrg-challenges], the authors identify the need for presenting programmable interfaces that accept instructions in a standards-supported manner for the Two-way Active Measurement Protocol (TWAMP)TWAMP protocol. More specifically, an excellent example in this case is traffic measurements, which are extensively used today to determine SLA adherence as well as debug and troubleshoot pain points in service delivery. TWAMP is both widely implemented by all established vendors and deployed by most global operators. However, TWAMP management and control today relies solely on diverse and proprietary tools provided by the respective vendors of the equipment. For large, virtualized, and dynamically instantiated infrastructures where network functions are placed according to orchestration algorithms proprietary mechanisms for managing TWAMP measurements have severe limitations. For example, today's TWAMP implementations are managed by vendor-specific, typically command-line interfaces (CLI), which can be scripted on a platform-by-platform basis. As a result, although the control and

test measurement protocols are standardized, their respective management is not. This hinders dramatically the possibility to integrate such deployed functionality in the SP-DevOps concept. In this particular case, recent efforts in the IPPM WG [I-D.cmzrjp-ippm-twamp-yang] aim to define a standard TWAMP data model and effectively increase the programmability of TWAMP deployments in the future.

Data center DevOps tools, such as those surveyed in [D4.1], developed proprietary methods for describing and interacting through interfaces with the managed infrastructure. Within certain communities, they became de-facto standards in the same way particular CLIs became de-facto standards for Internet professionals. Although open-source components and a strong community involvement exists, the diversity of the new languages and interfaces creates a burden for both vendors in terms of choosing which ones to prioritize for support, and then developing the functionality and operators that determine what fits best for the requirements of their systems.

11. DevOps Performance Metrics

Defining a set of metrics that are used as performance indicators is important for service providers to ensure the successful deployment and operation of a service in the software-defined telecom infrastructure.

We identify three types of considerations that are particularly relevant for these metrics: 1) technical considerations directly related to the service provided, 2) process-related considerations regarding the deployment, maintenance and troubleshooting of the service, i.e. concerning the operation of VNFs, and 3) cost-related considerations associated to the benefits from using a Software-Defined Telecom Infrastructure.

First, technical performance metrics shall be service-dependent/-oriented and may address inter-alia service performance in terms of delay, throughput, congestion, energy consumption, availability, etc. Acceptable performance levels should be mapped to SLAs and the requirements of the service users. Metrics in this category were defined in IETF working groups and other standardization organizations with responsibility over particular service or infrastructure descriptions.

Second, process-related metrics shall serve a wider perspective in the sense that they shall be applicable for multiple types of

services. For instance, process-related metrics may include: number of probes for end-to-end QoS monitoring, number of on-site interventions, number of unused alarms, number of configuration mistakes, incident/trouble delay resolution, delay between service order and deliver, or number of self-care operations.

Third, cost-related metrics shall be used to monitor and assess the benefit of employing SDI compared to the usage of legacy hardware infrastructure with respect to operational costs, e.g. possible man-hours reductions, elimination of deployment and configuration mistakes, etc.

Finally, identifying a number of highly relevant metrics for DevOps and especially monitoring and measuring them is highly challenging because of the amount and availability of data sources that could be aggregated within one such metric, e.g. calculation of human intervention, or secret aspects of costs.

12. Security Considerations

TBD

13. IANA Considerations

This memo includes no request to IANA.

14. References

14.1. Informative References

[NFVMANO] ETSI, "Network Function Virtualization (NFV) Management and Orchestration V0.6.1 (draft)", Jul. 2014

[I-D.aldrin-sfc-oam-framework] S. Aldrin, R. Pignataro, N. Akiya. "Service Function Chaining Operations, Administration and Maintenance Framework", draft-aldrin-sfc-oam-framework-02, (work in progress), July 2015.

[I-D.lee-sfc-verification] S. Lee and M. Shin. "Service Function Chaining Verification", draft-lee-sfc-verification-00, (work in progress), February 2014.

[RFC7426] E. Haleplidis (Ed.), K. Pentikousis (Ed.), S. Denazis, J. Hadi Salim, D. Meyer, and O. Koufopavlou, "Software Defined Networking (SDN): Layers and Architecture Terminology", RFC 7426, January 2015

[RFC7149] M. Boucadair and C Jaquenet. "Software-Defined Networking: A Perspective from within a Service Provider Environment", RFC 7149, March 2014.

[TR228] TMForum Gap Analysis Related to MANO Work. TR228, May 2014

[I-D.unify-nfvrg-challenges] R. Szabo et al. "Unifying Carrier and Cloud Networks: Problem Statement and Challenges", draft-unify-nfvrg-challenges-03 (work in progress), October 2016

[I-D.cmzrjp-ippm-twamp-yang] Civil, R., Morton, A., Zheng, L., Rahman, R., Jethanandani, M., and K. Pentikousis, "Two-Way Active Measurement Protocol (TWAMP) Data Model", draft-cmzrjp-ippm-twamp-yang-02 (work in progress), October 2015.

[D4.1] W. John et al. D4.1 Initial requirements for the SP-DevOps concept, universal node capabilities and proposed tools, August 2014.

[SDNsurvey] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. Esteve Rothenberg, S. Azodolmolky, S. Uhlig. "Software-Defined Networking: A Comprehensive Survey." To appear in proceedings of the IEEE, 2015.

[DevOpsP] "DevOps, the IBM Approach" 2013. [Online].

[Y1564] ITU-R Recommendation Y.1564: Ethernet service activation test methodology, March 2011

[CAP] E. Brewer, "CAP twelve years later: How the "rules" have changed", IEEE Computer, vol.45, no.2, pp.23,29, Feb. 2012.

[H2014] N. Handigol, B. Heller, V. Jeyakumar, D. Mazieres, N. McKeown; "I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks", In Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pp.71-95

[W2011] A. Wundsam, D. Levin, S. Seetharaman, A. Feldmann; "OFRewind: Enabling Record and Replay Troubleshooting for Networks". In Proceedings of the Usenix Anual Technical Conference (Usenix ATC '11), pp 327-340

[S2010] E. Al-Shaer and S. Al-Haj. "FlowChecker: configuration analysis and verification of federated Openflow infrastructures" In Proceedings of the 3rd ACM workshop on Assurable and usable security configuration (SafeConfig '10). Pp. 37-44

[OSandS] S. Wright, D. Druta, "Open Source and Standards: The Role of Open Source in the Dialogue between Research and Standardization" Globecom Workshops (GC Wkshps), 2014 , pp.650,655, 8-12 Dec. 2014

[C2015] CFEngine. Online: http://cfengine.com/product/what-is-cfengine/, retrieved Sep 23, 2015.

[P2015] Puppet. Online: http://puppetlabs.com/puppet/what-is-puppet, retrieved Sep 23, 2015.

[A2015] Ansible. Online: http://docs.ansible.com/ , retrieved Sep 23, 2015.

[AK2015] Apache Kafka. Online: http://kafka.apache.org/documentation.html, retrieved Sep 23, 2015.

[S2015] Splunk. Online: http://www.splunk.com/en_us/products/splunk-light.html , retrieved Sep 23, 2015.

[K2014] J. Kreps. Benchmarking Apache Kafka: 2 Million Writes Per Second (On Three Cheap Machines). Online: https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines, retrieved Sep 23, 2015.

[R2015] RabbitMQ. Online: https://www.rabbitmq.com/ , retrieved Oct 13, 2015

[Z2015] ZeroMQ. Online: http://zeromq.org/ , retrieved Oct 13, 2015

15. Contributors

W. John (Ericsson), J. Kim (Deutsche Telekom), S. Sharma (iMinds)

16. Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement no. 619609 - the UNIFY project. The views expressed here are those of the authors only. The European Commission is not liable for any use that may be made of the information in this document.

We would like to thank in particular the UNIFY WP4 contributors, the internal reviewers of the UNIFY WP4 deliverables and Russ White and Ramki Krishnan for their suggestions.

This document was prepared using 2-Word-v2.0.template.dot.

17. Authors' Addresses

Catalin Meirosu
Ericsson Research
S-16480 Stockholm, Sweden
Email: catalin.meirosu@ericsson.com

Antonio Manzalini
Telecom Italia
Via Reiss Romoli, 274
10148 - Torino, Italy
Email: antonio.manzalini@telecomitalia.it

Juhoon Kim
Deutsche Telekom AG
Winterfeldtstr. 21
10781 Berlin, Germany
Email: J.Kim@telekom.de

Rebecca Steinert
SICS Swedish ICT AB
Box 1263, SE-16429 Kista, Sweden
Email: rebste@sics.se

Sachin Sharma
Ghent University-iMinds
Research group IBCN - Department of Information Technology
Zuiderpoort Office Park, Blok C0
Gaston Crommenlaan 8 bus 201
B-9050 Gent, Belgium
Email: sachin.sharma@intec.ugent.be

Guido Marchetto
Politecnico di Torino
Corso Duca degli Abruzzi 24
10129 - Torino, Italy
Email: guido.marchetto@polito.it

Ioanna Papafili
Hellenic Telecommunications Organization
Measurements and Wireless Technologies Section
Laboratories and New Technologies Division
2, Spartis & Pelika str., Maroussi,
GR-15122, Attica, Greece
Buidling E, Office 102

Email: iopapafi@oteresearch.gr

Kostas Pentikousis
EICT GmbH
Torgauer Strasse 12-15
Berlin 10829
Germany
Email: k.pentikousis@eict.de

Steven Wright
AT&T Services Inc.
1057 Lenox Park Blvd NE, STE 4D28
Atlanta, GA 30319
USA
Email: sw3588@att.com

Wolfgang John
Ericsson Research
S-16480 Stockholm, Sweden
Email: wolfgang.john@ericsson.com