BMWG
Internet-Draft
Intended status: Informational
Expires: April 23, 2021

R. Rosa, Ed.
C. Rothenberg
UNICAMP
M. Peuster
H. Karl
UPB

October 20, 2020

Methodology for VNF Benchmarking Automation
draft-rosa-bmwg-vnfbench-06

Abstract

This document describes a common methodology for the automated benchmarking of Virtualized Network Functions (VNFs) executed on general-purpose hardware. Specific cases of automated benchmarking methodologies for particular VNFs can be derived from this document. An open source reference implementation is reported as running code embodiment of the proposed, automated benchmarking methodology.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 23, 2021.

Table of Contents

1. Introduction

In [RFC8172] the Benchmarking Methodology Working Group (BMWG) presented considerations for benchmarking of VNFs and their infrastructure, similar to the motivation given, the following aspects reinforce and justify the need for VNF benchmarking: (i) pre-deployment infrastructure dimensioning to realize associated VNF performance profiles; (ii) comparison factor with physical network functions; (iii) and output results for analytical VNF development.

Even if many methodologies the BMWG already describes, e.g., self-contained black-box benchmarking, can be applied to VNF benchmarking scenarios, further considerations have to be made. This is because VNFs, which are software components, might not have strict and clear execution boundaries and depend on underlying virtualization environment parameters as well as management and orchestration decisions [ETS14a].

Different enabling technologies advent of Software Defined Networking (SDN) and Network Functions Virtualization (NFV) have propitiated the disaggregation of VNFs and benchmarking tools, turning their Application Programming Interfaces (APIs) open and programmable. This process have occurred mostly by: (i) the decoupling of network function's control and data planes; (ii) the development of VNFs as multi-layer and distributed software components; (iii) and the existence of multiple underlying hardware abstractions to be utilized by VNFs.

Utilizing SDN and NFV enabling technologies, a diversity of benchmarking tools have been created to facilitate the active stimulus and the passive monitoring of a VNF via diverse software abstraction layers, propitiating a wide variety of abstractions for benchmarking mechanisms in the formulation of a VNF benchmarking methodology. In this manner of establishing the disaggregation of a VNF benchmarking setup, the abstracted VNF benchmarking mechanisms can be programmable, enabling the execution of their underlying technologies by the means of well defined parameters and producing a report with standardized metrics.

Turning programmable the execution of a VNF benchmarking methodology enables a richer apparatus for the benchmarking of a VNF and consequently facilitates the high-fidelity assessment of a VNF behaviour. Estimating the behaviour of a VNF depends on three correlated factors:

Internal configuration: Each use case of the VNF might define specific settings for it to work properly, and even each VNF might dispose of specific settings to be configured.

Hardware and software execution environment: A myriad of capabilities offered by execution environments might match in a large diversity of manners the possible internal software arrangements that each VNF might be programmable.

Network workload specificities: Depending on the use case, a VNF might be placed in different settings, operating under varied traffic profiles and in demand of a specific performance behavior.

The role of a VNF benchmarking methodology consists in defining how to tackle the diversity of settings imposed by the above enlisted factors in order to extract performance metrics associated with particular VNF packet processing behaviors. The sample space of testing such diversity of settings can be extensively large, turning manual benchmarking experiments prohibitively expensive. Indeed, portability as an intrinsic characteristic of VNFs allows them to be deployed in multiple execution environments, enabling benchmarking setups in a myriad of settings. Thus, the establishment of a methodology for VNF benchmarking automation detains utter importance.

Accordingly, can and should the flexible, software-based nature of VNFs be exploited to fully automate the entire benchmarking methodology end-to-end. This is an inherent need to align VNF benchmarking with the agile methods enabled by the concept of Network Functions Virtualization (NFV) [ETS14e]. More specifically it allows: (i) the development of agile performance-focused DevOps methodologies for Continuous Integration and Delivery (CI/CD) of VNFs; (ii) the creation of on-demand VNF test descriptors for upcoming execution environments; (iii) the path for precise-analytics of automated catalogues of VNF performance profiles; (iv) and run-time mechanisms to assist VNF lifecycle orchestration/management workflows, e.g., automated resource dimensioning based on benchmarking insights.

2. Terminology

Common benchmarking terminology contained in this document is derived from [RFC1242]. The reader is assumed to be familiar with the terminology as defined in the European Telecommunications Standards Institute (ETSI) NFV document [ETS14b]. Some of these terms, and others commonly used in this document, are defined below.

NFV: Network Function Virtualization - the principle of separating network functions from the hardware they run on by using virtual hardware abstraction.

VNF: Virtualized Network Function - a software-based network function. A VNF can be either represented by a single entity or be composed by a set of smaller, interconnected software components, called VNF components (VNFCs) [ETS14d]. Those VNFs are also called composed VNFs.

VNFC: Virtualized Network Function Component - a software component that implements (parts of) the VNF functionality. A VNF can consist of a single VNFC or multiple, interconnected VNFCs [ETS14d]

VNFD: Virtualised Network Function Descriptor - configuration template that describes a VNF in terms of its deployment and operational behaviour, and is used in the process of VNF on-boarding and managing the life cycle of a VNF instance.

NS: Network Service - a collection of interconnected VNFs forming a end-to-end service. The interconnection is often done using chaining of functions.

VNF Benchmarking Descriptor (VNF-BD) -- contains all the definitions and requirements to deploy, configure, execute, and reproduce VNF benchmarking tests. A VNF-BD is defined by the developer of a VNF benchmarking methodology and serve as input to the execution of an automated benchmarking methodology.

VNF Performance Profile (VNF-PP) -- in a well defined structure contains all the measured metrics resulting from the execution of automated VNF benchmarking tests defined by a specific VNF-BD. Additionally, it might also contain additional recordings of configuration parameters used during the execution of the benchmarking setup.

VNF Benchmarking Report (VNF-BR) -- contains all the definition of the inputs and outputs of an automated VNF benchmarking methodology. The inputs define the necessary VNF-BD and a respective list of variables referencing the VNF-BD fields that must be utilized to define the sample space of the VNF benchmarking settings. The outputs consist of a list of entries, each one contains one of the combinations of the sampled variables from the inputs, the input VNF-BD parsed with such combination of variables, and the obtained VNF-PP resulting from the automated realization of the parsed VNF-BD. A VNF-BR might contain the settings definitions of the orchestrator platform that realizes

the instantiation of the benchmarking setup to enable the VNF-BD fullfilment.

3. Scope

This document assumes VNFs as black boxes when defining their benchmarking methodologies. White box approaches are assumed and analysed as a particular case under the proper considerations of internal VNF instrumentation, later discussed in this document.

This document outlines a methodology for VNF benchmarking, specifically addressing its automation, without limiting the automated process to a specific benchmarking case or infrastructure. The document addresses state-of-the-art work on VNF benchmarking from scientific publications and current developments in other standardization bodies (e.g., [ETS14c], [ETS19f] and [RFC8204]) wherever possible.

Whenever utilizing the specifications of this document, a particular automated VNF benchmarking methodology must be described in a clear and objective manner following four basic principles:

- Comparability: The output of a benchmarking test shall be simple to understand and process, in a human-readable format, coherent, and easily reusable (e.g., inputs for analytic applications).

- Repeatability: A benchmarking setup shall be comprehensively defined through a flexible design model that can be interpreted and executed by the testing platform repeatedly but supporting customization.

- Configurability: Open interfaces and extensible messaging models shall be available between benchmarking components for flexible composition of a benchmarking test descriptor and environment configurations.

- Interoperability: A benchmarking test shall be ported to different environments, using lightweight components whenever possible.

4. Considerations

VNF benchmarking considerations are defined in [RFC8172]. Additionally, VNF pre-deployment testing considerations are well explored in [ETS14c]. Further, ETSI provides test specifications for networking benchmarks and measurement methods for NFV infrastructure in [ETS19f], which complements the presented work on VNF benchmarking methodologies.

4.1. VNF Assessment Methods

Following ETSI's model in [ETS14c], we distinguish three methods for a VNF evaluation:

Benchmarking: Where parameters (e.g., CPU, memory, storage) are provided and the corresponding performance metrics (e.g., latency, throughput) are obtained. Note, such evaluations might create multiple reports, for example, with minimal latency or maximum throughput results.

Verification: Both parameters and performance metrics are provided and a stimulus verifies if the given association is correct or not.

Dimensioning: Performance metrics are provided and the corresponding parameters obtained. Note, multiple deployments may be required, or if possible, underlying allocated resources need to be dynamically altered.

Note: Verification and Dimensioning can be reduced to Benchmarking.

4.2. Benchmarking Stages

The realization of an automated benchmarking methodology can be divided into three stages:

Trial: Is a single process or iteration to obtain VNF performance metrics from benchmarking measurements. A Test MUST always run multiple Trials to get statistical confidence about the obtained measurements.

Test: Defines unique structural and functional parameters (e.g., configurations, resource assignment) for benchmarked components to perform one or multiple Trials. Each Test must be executed following a particular benchmarking scenario composed by a Method. Proper measures must be taken to ensure statistical validity (e.g., independence across Trials of generated load patterns).

Method: Consists of one or more Tests to benchmark a VNF. A Method can explicitly list ranges of parameter values for the configuration of a benchmarking scenario and its components. Each value of such a range is to be realized in a Test. I.e., Methods can define parameter studies.

4.3. Architectural Framework

A VNF benchmarking architectural framework, shown in Figure 1, establishes the disposal of essential components and control interfaces, explained below, that realize the automation of a VNF benchmarking methodology.

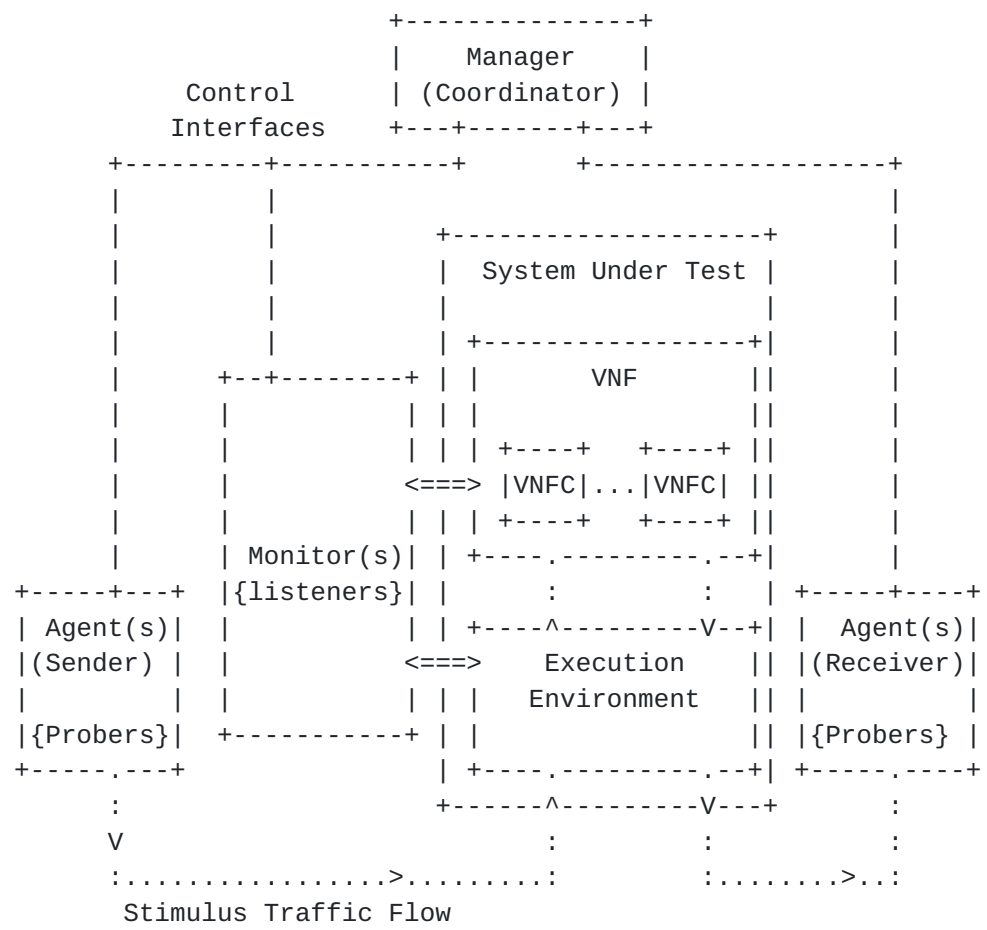

Figure 1: A VNF Benchmarking Architectural Framework

Virtualized Network Function (VNF) -- consists of one or more software components, so called VNF components (VNFC), adequate for performing a network function according to allocated virtual resources and satisfied requirements in an execution environment. A VNF can demand particular settings for benchmarking specifications, demonstrating variable performance based on available virtual resource parameters and configured enhancements targeting specific technologies (e.g., NUMA, SR-IOV, CPU-Pinning).

Execution Environment -- defines a virtualized and controlled composition of capabilities necessary for the execution of a VNF. An execution environment stands as a general purpose level of virtualization with abstracted resources available for one or more VNFs. It can also define specific technology qualifications, incurring in viable settings for enhancing the performance of VNFs, satisfying their particular enhancement requirements. An execution environment must be defined with the proper virtualization technologies feasible for the allocation of a VNF. The means to programmatically control the execution environment capabilities must be well defined for its life cycle management.

Agent (Active Prospection) -- executes active stimulus using probers, to benchmark and collect network and system performance metrics. A single Agent can perform localized benchmarks in execution environments (e.g., stress tests on CPU, memory, storage Input/Output) or can generate stimulus traffic and the other end be the VNF itself where, for example, one-way latency is evaluated. The interaction among two or more Agents enable the generation and collection of end-to-end metrics (e.g., frame loss rate, latency) measured from stimulus traffic flowing through a VNF. An Agent can be defined by a physical or virtual network function, and it must provide programmable interfaces for its life cycle management.

Prober -- defines an abstraction layer for a software or hardware tool able to generate stimulus traffic to a VNF or perform stress tests on execution environments. Probers might be specific or generic to an execution environment or a VNF. For an Agent, a Prober must provide programmable interfaces for its life cycle management, e.g., configuration of operational parameters, execution of stilumus, parsing of extracted metrics, and debugging options. Specific Probers might be developed to abstract and to realize the description of particular VNF benchmarking methodologies.

Monitor (Passive Prospection) -- when possible is instantiated inside the System Under Test, VNF and/or execution environment, to perform the passive monitoring, using Listeners, for the extraction of metrics while Agents` stimuli takes place. Monitors observe particular properties according to the execution environment and VNF capabilities, i.e., exposed passive monitoring interfaces. Multiple Listeners can be executed at once in synchrony with a Prober' stimulus on a SUT. A Monitor can be defined as a virtualized network function, and it must provide programmable interfaces for its life cycle management.

Listener -- defines one or more software interfaces for the extraction of metrics monitored in a target VNF and/or execution environment. A Listener must provide programmable interfaces for its life cycle management workflows, e.g., configuration of operational parameters, execution of passive monitoring captures, parsing of extracted metrics, and debugging options (also see [ETS19g]). Varied methods of passive performance monitoring might be implemented as a Listener, depending on the interfaces exposed by the VNF and/or the execution environment.

Manager -- performs (i) the discovery of available Agents and Monitors and their respective features (i.e., available Probers/ Listeners and their execution environment capabilities), (ii) the coordination and synchronization of activities of Agents and Monitors to perform a benchmarking Test, (iii) the collection, processing and aggregation of all VNF benchmarking (active and passive) metrics, which correlates the characteristics of the VNF traffic stimuli and the, possible, SUT monitoring. A Manager executes the main configuration, operation, and management actions to deliver the VNF benchmarking metrics. Hence, it detains interfaces open for users interact with the whole benchmarking framework, realizing, for instance, the retrival of the framework characteristics (e.g., available benchmarking components and their probers/listeners), the coordination of benchmarking tests, the processing and the retrival of benchmarking metrics, among other operational and management functionalities. A Manager can be defined as a physical or virtualized network function, and it must provide programmable interfaces for its life cycle management.

4.4. Scenarios

A scenario, as well referred as a benchmarking setup, consists of the actual instantiation of physical and/or virtual components of a "VNF Benchmarking Architectural Framework" needed to habilitate the execution of an automated VNF benchmarking methodology. The following considerations hold for a scenario:

- Not all components are mandatory for a Test, possible to be disposed in varied setups.

- Components can be aggregated in a single entity and be defined as black or white boxes. For instance, Manager and Agents could jointly define one hardware or software entity to perform a VNF benchmarking Test.

- Monitor can be defined by multiple instances of distributed software components, each one addressing one or more VNF or execution environment monitoring interfaces.

- Agents can be disposed in varied topology setups, included the possibility of multiple input and output ports of a VNF being directly connected each in one Agent.

- All benchmarking components defined in a scenario must perform the synchronization of clocks.

4.5. Phases of a Benchmarking Test

In general, an automated benchmarking methodology must execute Tests repeatedly so it must capture the relevant causes of the performance variability of a VNF. To dissect a VNF benchmarking Test, in the sections that follow a set of benchmarking phases are categorized defining generic operations that may be automated. When executing an automated VNF benchmarking methodology, all the influencing aspects on the performance of a VNF must be carefully analyzed and comprehensively reported in each automated phase of a benchmarking Test.

4.5.1. Phase I: Deployment

The placement (i.e., assignment and allocation of resources) and the interconnection, physical and/or virtual, of network function(s) and benchmarking components can be realized by orchestration platforms (e.g., OpenStack, Kubernetes, Open Source MANO). In automated manners, the realization of a benchmarking scenario through those means usually rely on network service templates (e.g., TOSCA, YANG, Heat, and Helm Charts). Such descriptors have to capture all relevant details of the execution environment to allow the benchmarking framework to correctly instantiate the SUT as well as helper functions required for a Test.

4.5.2. Phase II: Configuration

The configuration of benchmarking components and VNFs (e.g., populate routing table, load PCAP source files in source of traffic stimulus) to execute the Test settings can be realized by programming interfaces in an automated way. In the scope of NFV, there might exist management interfaces to control a VNF during a benchmarking Test. Likewise, infrastructure or orchestration components can establish the proper configuration of an execution environment to realize all the capabilities enabling the description of the benchmarking Test. Each configuration registry, its deployment

timestamp and target, must all be contained in the report of a VNF benchmarking Test.

4.5.3. Phase III: Execution

In the execution of a benchmarking Test, the VNF configuration can be programmed to be changed by itself or by a VNF management platform. It means that during a Trial execution, particular behaviors of a VNF can be automatically triggered, e.g., auto-scaling of its internal components. Those must be captured in the detailed procedures of the VNF execution and its performance report. I.e., the execution of a Trial can determine arrangements of internal states inside a VNF, which can interfere in observed benchmarking metrics. For instance, in a particular benchmarking case where the monitoring measurements of the VNF and/or execution environment are available for extraction, comparison Tests must be run to verify if the monitoring of the VNF and/or execution environment can impact the VNF performance metrics.

4.5.4. Phase IV: Result

The result of a VNF benchmarking Test might contain generic metrics (e.g., CPU and memory consumption) and VNF-specific traffic processing metrics (e.g., transactions or throughput), which can be stored and processed in generic or specific ways (e.g., by statistics or machine learning algorithms). More details about possible metrics and the corresponding capturing methods can be found in [ETS19g]. If automated procedures are applied over the generation of a benchmarking Test result, those must be explained in the result itself, jointly with their input raw measurements and output processed data. For instance, any algorithm used in the generation of processed metrics must be disclosed in the Test result.

5. Methodology

The execution of an automated benchmarking methodology consists in elaborating a VNF Benchmarking Report, its inputs and outputs. The inputs part of a VNF-BR must be written by a VNF benchmarking tester. When the VNF-BR, with its inputs fulfilled, is requested from the Manager component of a implementation of the "VNF Benchmarking Architectural Framework", the Manager must utilize the inputs part to obtain the outputs part of the VNF-BR, addressing the execution of the automated benchmarking methodology as defined in Section 5.4.

The flow of information in the execution of an automated benchmarking methodology can be represented by the YANG modules defined by this document. The sections that follow present an overview of such modules.

5.1. VNF Benchmarking Descriptor (VNF-BD)

VNF Benchmarking Descriptor (VNF-BD) -- an artifact that specifies how to realize the Test(s) and Trial(s) of an automated VNF benchmarking methodology in order to obtain a VNF Performance Profile. The specification includes structural and functional instructions and variable parameters at different abstraction levels, such as the topology of the benchmarking scenario, and the execution parameters of prober(s)/listener(s) in the required Agent(s)/Monitor(s). A VNF-BD may be specific to a VNF or applicable to several VNF types.

More specifically, a VNF-BD is defined by a scenario and its proceedings. The scenario defines nodes (i.e., benchmarking components) and links interconnecting them, a topology that must be instantiated in order to execute the VNF-BD proceedings. The proceedings contain the specification of the required Agent(s) and Monitor(s) needed in the scenario nodes. Detailed in each Agent/ Monitor follows the specification of the Prober(s)/Listener(s) required for the execution of the Tests, and in the details of each Prober/Listener follows the specification of its execution parameters. In the header of a VNF-BD is specified the number of Tests and Trials that a Manager must run them. Each Test realizes a unique instantiation of the scenario, while each Trial realizes a unique execution of the proceedings in the instantiated scenario of a Test. The VNF-BD YANG module is presented in Section 10.1.

5.2. VNF Performance Profile (VNF-PP)

VNF Performance Profile (VNF-PP) -- an output artifact of a VNF-BD execution performed by a Manager component. It contains all the metrics from Monitor(s) and/or Agent(s) components after realizing the execution of the Prober(s) and/or the Listener(s) proceedings, specified in its corresponding VNF-BD. Metrics are logically grouped according to the execution of the Trial(s) and Test(s) defined by a VNF-BD. A VNF-PP is specifically associated with a unique VNF-BD.

More specifically, a VNF-PP is defined by a structure that allows benchmarking results to be presented in a logical and unified format. A VNF-PP report is the result of an unique Test, while its content, the so called snapshot(s), each containing the results of the execution of a single Trial. Each snapshot is built by a single Agent or Monitor. A snapshot contains evaluation(s), each one being the output of the execution of a single Prober or Listener. An evaluation contains one or more metrics. In summary, a VNF-PP aggregates the results from reports (i.e., the Test(s)); a report aggregates Agent(s) and Monitor(s) results (i.e., the Trial(s)); a snapshot aggregates Prober(s) or Listener(s) results; and an

evaluation aggregates metrics. The VNF-PP YANG module is presented in Section 10.2.

5.3. VNF Benchmarking Report (VNF-BR)

VNF Benchmarking Report (VNF-BR) -- the core artifact of an automated VNF benchmarking methodology consisted of three parts: a header, inputs and output. The header refers to the VNF-BR description items (e.g., author, version, name), the description of the target SUT (e.g., the VNF version, release, name), and the environment settings specifying the parameters needed to instantiate the benchmarking scenario via an orchestration platform. The inputs contain the definitions needed to execute the automated benchmarking methodology of the target SUT, a VNF-BD and its variables settings. The outputs contain the results of the execution of the inputs, a list of entries, each one containing a VNF-BD filled with one of the combinations of the input variables settings, and the obtained VNF-PP reported after the execution of the Test(s) and Trial(s) of the parsed VNF-BD. The process of utilizing the VNF-BR inputs to generate its outputs concerns the realization of an automated VNF benchmarking methodology, explained in details in Section 5.4.2. The VNF-BR YANG module is presented in Section 10.3.

In details, each one of the variables in the inputs part of a VNF-BR is defined by: a name (the actual name of the variable); a path (the YANG path of the variable in the input VNF-BD); a type (the type of the values, such as string, int, float, etc); class (one of: stimulus, resource, configuration); and values (a list of the variable actual values). The values of all the variables must be combined all-by-all, generating a list containing the whole sample space of variables settings that must be used to create the VNF-BD instances. A VNF-BD instance is defined as the result of the parsing of one of those combinations of input variables into the VNF-BD of the VNF-BR inputs. The parsing takes place when the variable path is utilized to set its value in the VNF-BD. Interatively, all the VNF-BD instances must have its Test(s) and Trial(s) executed to generate its corresponding VNF-PP. After all the VNF-BD instances had their VNF-PP accomplished, the realization of the whole automated VNF benchmarking methodology is complete, fulfilling the outputs part of the VNF-BR as shown in Figure 2.

5.4. Procedures

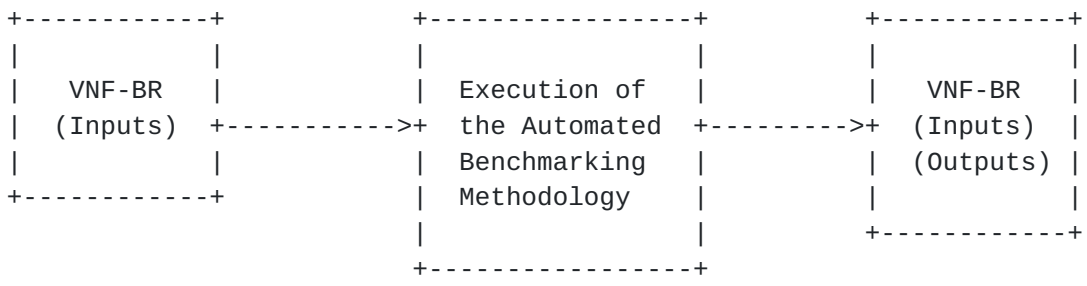

Figure 2: VNF benchmarking process inputs and outputs

The methodology for VNF benchmarking automation encompasses the process defined in Figure 2, i.e., the procedures that utilize the inputs part to obtain the outputs part of a VNF-BR. This section details the procedures that realize such process.

5.4.1. Plan

The plan of an automated VNF benchmarking methodology consists in the definition of all the header and the inputs part of a VNF-BR, the artifacts to be utilized by the realization of the methodology, and the establishment of the execution environment where the methodology takes place. The topics below contain the details of such planning.

1. The writing of a VNF-BD must be done utilizing the VNF-BD YANG module Section 10.1. A VNF-BD composition must determine the scenario and the proceedings. The VNF-BD must be added to the inputs part of an instance of the VNF-BR YANG model.

2. All the variables in the inputs part of a VNF-BR must be defined. Each variable must contain all its fields fullfiled according to the VNF-BR YANG module Section 10.3.

3. All the software artifacts needed for the instantiation of the VNF-BD scenario must be made and turn available for the execution of the Test(s) and Trial(s). The artifacts include the definition of software components that realize the role of the functional components of the Benchmarking Architectural Framework, i.e., the Manager, the Agent and the Monitor and their respective Probers and Listeners.

4. The header of the VNF-BR instance must be written, stating the VNF-BR description items, the specification of the SUT settings, and the definition of the environment parameters, feasible for the instantiation of the VNF-BD scenario when executing the automated VNF benchmarking methodology.

5. The execution environment needed for a VNF-BD scenario must be prepared to be utilized by an orchestration platform to automate instantiation of the scenario nodes and links needed for the execution of a Test. The orchestration platform interface parameters must be referenced in the VNF-BR header. The orchestration platform must have access to the software artifacts that are referenced in the VNF-BD scenario to be able to manage their life cycle.

6. The Manager component must be instantiated, the execution environment must be turned available, and the orchestration platform must have accesss to the execution environment and the software artifacts that are referenced in the scenario of the VNF-BD in the inputs part of the VNF-BR.

5.4.2. Realization

Accomplished all the planning procedures, the process of the realization of the automated benchmarking methodology must be realized as the following topics describe.

1. The realization of the benchmarking procedures starts when the VNF-BR composed in the planning procedures is submitted to the Manager component. It triggers the automated execution of the benchmarking methodology defined by the inputs part of the VNF-BR.

2. Manager computes all the combinations of values from the lists of inputs in the VNF-BD, part of the submitted VNF-BR. Each combination of variables are used to define a Test. The VNF-BD submitted serves as a template for each combination of variables. Each parsing of each combination of variables by the VNF-BD template creates a so called VNF-BD instance. The Manager must iterate through all the VNF-BD instances to finish the whole set of Tests defined by all the combinations of variables and their respective parsed VNF-BD. The Manager iterates through the following steps until all the Tests are accomplished.

3. The Manager must interface an orchestration platform to realize the automated instantiation of the deployment scenario defined by a VNF-BD instance (i.e., a Test). To perform such step, The Manager might interface a management function responsible to properly parse the deployment scenario specifications into the orchestration platform interface format. The environment specifications of the VNF-BR header provide the guidelines to interface the orchestration platform. The orchestration platform must deploy the scenario requested by the Manager, assuring the requirements and policies specified on it. In addition, the orchestration platform must acknowledge the deployed scenario to

the Manager specifying the management interfaces of the VNF SUT and the other components in the running instances for the benchmarking scenario. Only when the scenario is correctly deployed the execution of the VNF-BD instance Test(s) and Trial(s) must ocurr, otherwise the whole execution of the VNF-BR must be aborted and an error message must be added to the VNF-BR outputs describing the problems that ocurred in the instantiation of the VNF-BD scenario. If the scenario is successfuly deployed, the VNF-BD Test proceedings can be executed.

4. Manager must interface Agent(s) and Monitor(s) via their management interfaces to require the execution of the VNF-BD proceedings, which consist in running the specified Probers and Listeners using the defined parameters, and retrieve their output metrics captured at the end of each Trial. Thus, a Trial conceives the execution of the proceedings of the VNF-BD instance. The number of Trials is defined in each VNF-BD instance. After the execution of all defined Trials the execution of a Test ends.

5. Output measurements from each obtained benchmarking Trials that compose a Test result must be collected by the Manager, until all the Tests are finished. Each set of collected measurements from each VNF-BD instance Trials and Tests must be used to elaborate a VNF-PP by the Manager component. The respective VNF-PP, its associated VNF-BD instance and its input variables compose one of the entries of the list of outputs of the VNF-BR. After all the list of combinations of input variables is explored to obtain the whole list of instances of VNF-BDs and elaborated VNF-PPs, the Manager component returns the original VNF-BR submitted to it, including the outputs part properly filled.

5.4.3. Summary

After the realization of an automated benchmarking methodology, some automated procedures can be performed to improve the quality and the utility of the obtained VNF-BR, as described in the following topics.

1. Archive the raw outputs contained in the VNF-BR, perform statistical analysis on it, or train machine learning models with the collected data.

2. Evaluate the analysis output to the detection of any possible cause-effect factors and/or intrinsic correlations in the VNF-BR outputs (e.g., outliers).

3. Review the inputs of a VNF-BR, VNF-BD and variables, and modify them to realize the proper extraction of the target VNF metrics based on the intended goal of the VNF benchmarking methodology

(e.g., throughput). Iterate in the previous steps until composing a stable and representative VNF-BR.

6. Particular Cases

As described in [RFC8172], VNF benchmarking might require to change and adapt existing benchmarking methodologies. More specifically, the following cases need to be considered.

6.1. Capacity

VNFs are usually deployed inside containers or VMs to build an abstraction layer between physical resources and the resources available to the VNF. According to [RFC8172], it may be more representative to design experiments in a way that the VMs hosting the VNFs are operating at maximum of 50% utilization and split the workload among several VMs, to mitigateside effects of overloaded VMs. Those cases are supported by the presented automation methodologies through VNF-BDs that enable direct control over the resource assignments and topology layouts used for a benchmarking experiment.

6.2. Redundancy

As a VNF might be composed of multiple components (VNFCs), there exist different schemas of redundancy where particular VNFCs would be in active or standby mode. For such cases, particular monitoring endpoints should be specified in VNF-BD so listeners can capture the relevant aspects of benchmarking when VNFCs would be in active/ standby modes. In this particular case, capturing the relevant aspects of internal functionalities of a VNF and its internal components provides important measurements to characterize the dynamics of a VNF, those must be reflected in its VNF-PP.

6.3. Isolation

One of the main challenges of NFV is to create isolation between VNFs. Benchmarking the quality of this isolation behavior can be achieved by Agents that take the role of a noisy neighbor, generating a particular workload in synchrony with a benchmarking procedure over a VNF. Adjustments of the Agent's noisy workload, frequency, virtualization level, among others, must be detailed in the VNF- BD.

6.4. Failure Handling

Hardware and software components will fail or have errors and thus trigger healing actions of the benchmarked VNFs (self-healing). Benchmarking procedures must also capture the dynamics of this VNF

behavior, e.g., if a container or VM restarts because the VNF software crashed. This results in offline periods that must be captured in the benchmarking reports, introducing additional metrics, e.g., max. time-to-heal. The presented concept, with a flexible VNF-PP structure to record arbitrary metrics, enables automation of this case.

6.5. Elasticity and Flexibility

Having software based network functions and the possibility of a VNF to be composed by multiple components (VNFCs), internal events of the VNF might trigger changes in VNF behavior, e.g.,activating functionalities associated with elasticity such as automated scaling. These state changes and triggers (e.g. the VNF's scaling state) must be captured in the benchmarking results (VNF-PP) to provide a detailed characterization of the VNF's performance behavior in different states.

6.6. Handling Configurations

As described in [RFC8172], does the sheer number of test conditions and configuration combinations create a challenge for VNF benchmarking. As suggested, machine readable output formats, as they are presented in this document, will allow automated benchmarking procedures to optimize the tested configurations. Approaches for this are, e.g., machine learning-based configuration space sub-sampling methods, such as [Peu-c].

6.7. White Box VNF

A benchmarking setup must be able to define scenarios with and without monitoring components inside the VNFs and/or the hosting container or VM. If no monitoring solution is available from within the VNFs, the benchmark is following the black-box concept. If, in contrast, those additional sources of information from within the VNF are available, VNF-PPs must be able to handle these additional VNF performance metrics.

7. Open Source Reference Implementation

Currently, technical motivating factors in favor of the automation of VNF benchmarking methodologies comprise: (i) the facility to run high-fidelity and commodity traffic generators by software; (ii) the existent means to construct synthetic traffic workloads purely by software (e.g., handcrafted pcap files); (iii) the increasing availability of datasets containing actual sources of production traffic able to be reproduced in benchmarking tests; (iv) the existence of a myriad of automating tools and open interfaces to

programmatically manage VNFs; (v) the varied set of orchestration platforms enabling the allocation of resources and instantition of VNFs through automated machineries based on well-defined templates; (vi) the ability to utilize a large tool set of software components to compose pipelines that mathematically analyze benchmarking metrics in automated ways.

In simple terms, the enlisted factors above justify that network softwarization enables the automation of VNF benchmarking methodologies. There exists an open source reference implementation that is built to demonstrate the concepts and methodology of this document in order to automate the benchmarking of Virtualized Network Functions.

7.1. Gym

The software, named Gym, is a framework for automated benchmarking of Virtualized Network Functions (VNFs). It was coded following the initial ideas presented in a 2015 scientific paper entitled "VBaaS: VNF Benchmark-as-a-Service" [Rosa-a]. Later, the evolved design and prototyping ideas were presented at IETF/IRTF meetings seeking impact into NFVRG and BMWG.

Gym was built to receive high-level test descriptors and execute them to extract VNFs profiles, containing measurements of performance metrics - especially to associate resources allocation (e.g., vCPU) with packet processing metrics (e.g., throughput) of VNFs. From the original research ideas [Rosa-a], such output profiles might be used by orchestrator functions to perform VNF lifecycle tasks (e.g., deployment, maintenance, tear-down).

In [Rosa-b] Gym was utilized to benchmark a decomposed IP Multimedia Subsystem VNF. And in [Rosa-c], a virtual switch (Open vSwitch - OVS) was the target VNF of Gym for the analysis of VNF benchmarking automation. Such articles validated Gym as a prominent open source reference implementation for VNF benchmarking tests. Such articles set important contributions as discussion of the lessons learned and the overall NFV performance testing landscape, included automation.

Gym stands as one open source reference implementation that realizes the VNF benchmarking methodologies presented in this document. Gym is released as open source tool under Apache 2.0 license [gym].

7.2. Related work: tng-bench

Another software that focuses on implementing a framework to benchmark VNFs is the "5GTANGO VNF/NS Benchmarking Framework" also called "tng-bench" (previously "son-profile") and was developed as

part of the two European Union H2020 projects SONATA NFV and 5GTANGO [tango]. Its initial ideas were presented in [Peu-a] and the system design of the end-to-end prototype was presented in [Peu-b].

Tng-bench aims to be a framework for the end-to-end automation of VNF benchmarking processes. Its goal is to automate the benchmarking process in such a way that VNF-PPs can be generated without further human interaction. This enables the integration of VNF benchmarking into continuous integration and continuous delivery (CI/CD) pipelines so that new VNF-PPs are generated on-the-fly for every new software version of a VNF. Those automatically generated VNF-PPs can then be bundled with the VNFs and serve as inputs for orchestration systems, fitting to the original research ideas presented in [Rosa-a] and [Peu-a].

Following the same high-level VNF testing purposes as Gym, namely: Comparability, repeatability, configurability, and interoperability, tng- bench specifically aims to explore description approaches for VNF benchmarking experiments. In [Peu-b] a prototype specification for VNF-BDs is presented which not only allows to specify generic, abstract VNF benchmarking experiments, it also allows to describe sets of parameter configurations to be tested during the benchmarking process, allowing the system to automatically execute complex parameter studies on the SUT, e.g., testing a VNF's performance under different CPU, memory, or software configurations.

Tng-bench was used to perform a set of initial benchmarking experiments using different VNFs, like a Squid proxy, an Nginx load balancer, and a Socat TCP relay in [Peu-b]. Those VNFs have not only been benchmarked in isolation, but also in combined setups in which up to three VNFs were chained one after each other. These experiments were used to test tng-bench for scenarios in which composed VNFs, consisting of multiple VNF components (VNFCs), have to be benchmarked. The presented results highlight the need to benchmark composed VNFs in end-to-end scenarios rather than only benchmark each individual component in isolation, to produce meaningful VNF- PPs for the complete VNF.

Tng-bench is actively developed and released as open source tool under Apache 2.0 license [tng-bench]. A larger set of example benchmarking results of various VNFs is available in [Peu-d].

8. Security Considerations

Benchmarking tests described in this document are limited to the performance characterization of VNFs in a lab environment with isolated network.

The benchmarking network topology will be an independent test setup and MUST NOT be connected to devices that may forward the test traffic into a production network, or misroute traffic to the test management network.

Special capabilities SHOULD NOT exist in the VNF benchmarking deployment scenario specifically for benchmarking purposes. Any implications for network security arising from the VNF benchmarking deployment scenario SHOULD be identical in the lab and in production networks.

9. IANA Considerations

This document registers one URI in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

```
URI: urn:ietf:params:xml:ns:yang:ietf-vnf-bd
Registrant Contact: The BMWG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-vnf-pp
Registrant Contact: The BMWG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-vnf-br
Registrant Contact: The BMWG of the IETF.
XML: N/A, the requested URI is an XML namespace.
```

Figure 3

This document registers three YANG modules in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the following registration is requested:

```
      name:         ietf-vnf-bd
      namespace:    urn:ietf:params:xml:ns:yang:ietf-vnf-bd
      prefix:       vnf-bd
      reference:    RFC CCCC

      name:         ietf-vnf-pp
      namespace:    urn:ietf:params:xml:ns:yang:ietf-vnf-pp
      prefix:       vnf-pp
      reference:    RFC CCCC

      name:         ietf-vnf-br
      namespace:    urn:ietf:params:xml:ns:yang:ietf-vnf-br
      prefix:       vnf-br
      reference:    RFC CCCC
```

Figure 4

10. YANG Modules

The following sections contain the YANG modules defined by this document.

10.1. VNF-Benchmarking Descriptor

```
module vnf-bd {
  namespace "urn:ietf:params:xml:ns:yang:vnf-bd";
  prefix "vnf-bd";

  organization "IETF/BMWG";
  contact "Raphael Vicente Rosa <raphaelvrosa@gmail.com>,
  Manuel Peuster <peuster@mail.uni-paderborn.de>";

  description "Yang module for a VNF Benchmarking
  Descriptor (VNF-BD).";

  revision "2019-08-13" {
    description "V0.3: Reviewed proceedings,
    tool - not VNF specific";
    reference "";
  }

  revision "2019-03-13" {
    description "V0.2: Reviewed role, policies, connection-points,
    lifecycle workflows, resources";
    reference "";
  }

  revision "2019-02-28" {
```

```
      description "V0.1: First release";
      reference "";
    }

    typedef workflows {
      type enumeration {
        enum create {
          description "When calling the create workflow.";
        }
        enum configure {
          description "When calling the configure workflow.";
        }
        enum start {
          description "When calling the start workflow.";
        }
        enum stop {
          description "When calling the stop workflow.";
        }
        enum delete {
          description "When calling the delete workflow.";
        }
        enum custom {
          description "When calling a custom workflow.";
        }
      }
      description "Defines basic life cycle workflows for a
      node in a scenario.";
    }

    grouping node_requirements {
      container resources {
        container cpu {
          leaf vcpus {
            type uint32;
            description "The number of cores to be allocated
            for a node.";
          }
          leaf cpu_bw {
            type string;
            description "The CPU bandwidth (CFS limit in 0.01-1.0)";
          }
          leaf pinning {
            type string;
            description "The list of CPU cores, separated by comma,
            that a node must be pinned to.";
          }
          description "The node CPU resources that must
          be allocated for a benchmarking Test.";
```

```
        }
        container memory {
          leaf size {
            type uint32;
            description "The memory allocation size.";
          }
          leaf unit {
            type string;
            description "The memory unit.";
          }
          description "The node memory resources
          that must be allocated for a benchmarking
          Test.";
        }
        container storage {
          leaf size {
            type uint32;
            description "The storage allocation size.";
          }
          leaf unit {
            type string;
            description "The storage unit.";
          }
          leaf volumes {
            type string;
            description "Volumes to be allocated by
            a node storage.
            A volume defines a mapping of an outside storage
            partition inside the node storage system.
            Volumes must be separated by comma and be defined
            using a colon to separate the node internal and external
            references of storage system paths.";
          }

          description "The node storage resources
          that must be allocated for a benchmarking Test.";
        }

        description "The set of resources that must be allocated
        for a node in a benchmarking Test.";
      }

      description "'The grouping determining the
      resource requirements for a node in a scenario.";
    }

    grouping connection_points {
      leaf id {
```

```
          type string;
          description "The connection-point
          unique identifier";
        }
        leaf interface {
          type string;
          description "The name of the node interface
          associated with the connection-point.";
        }
        leaf type {
          type string;
          description "The type of the network the
          connection-point interface is attached to.";
        }
        leaf address {
          type string;
          description "The Network address of the
          connection-point. It can be specified as a
          Ethernet MAC address, a IPv4 address or an IPv6 address.";
        }
        description "A connections-point of a node.";
      }

      grouping nodes {
        leaf id {
          type string;
          description "The unique identifier of a node
          in a scenario.";
        }
        leaf type {
          type string;
          description "The type of a node.";
        }
        leaf image {
          type string;
          description "The name of the image to be used to instantiate
          a node.";
        }
        leaf format {
          type string;
          description "The node format (e.g., container, process, VM).";
        }
        leaf role {
          type string;
          description "The role of the node in the Test scenario.
          The role must be one of: manager, agent, monitor, sut.";
        }
```

```
      uses node_requirements;

      list connection_points {
        key "id";
        uses connection_points;
        description "The list of connection points of a node.";
      }

      list relationships {
        key "name";
        leaf name {
          type string;
          description "Name of the relationship.";
        }
        leaf type {
          type string;
          description "Type of the relationship.";
        }
        leaf target {
          type string;
          description "Target of the relationship.";
        }

        description "Relationship of a node with the other
        scenario components.";
      }


      list lifecycle {
        key "workflow";
        leaf workflow {
          type workflows;
          description "The type of the Workflow.";
        }
        leaf name {
          type string;
          description "The workflow name.";
        }

        list parameters {
          key "input";
          leaf input {
            type string;
            description "The name of the parameter.";
          }
          leaf value {
            type string;
            description "The value of the parameter";
```

```
          }

          description "The list of parameters to be
          applied to the node workflow.";
        }

        leaf-list implementation {
          type string;
          description "The workflow implementation.";
        }

        description "The life cycle workflows to be
        applied to this node.";
      }

      description "The specification of a node to be used
      in a scenario for a benchmarking Test.";
    }

    grouping link {
      leaf id {
        type string;
        description "The link unique identifier.";
      }
      leaf name {
        type string;
        description "The name of the link.";
      }
      leaf type {
          type string;
          description "The type of the link.";
      }
      leaf network {
          type string;
          description "The network the link belongs to.";
      }
      leaf-list connection_points {
          type leafref {
              path "../../nodes/connection_points/id";
          }
          description "Reference to the connection points of nodes
          the link is adjacent.";
      }
      description "A link between nodes in a scenario.";
    }

    grouping scenario {
      list nodes {
```

```
        key "id";
        uses nodes;
        description "The list of nodes that must be
        instantiated in a scenario in order to enable
        a benchmarking Test.";
      }

      list links {
        key "id";
        uses link;
        description "The list of links among nodes that must be
        instantiated in a scenario in order to enable
        a benchmarking Test.";
      }

      list policies {
        key "name";
        leaf name {
          type string;
          description "The name of the policy.";
        }
        leaf type {
          type string;
          description "The type of the policy";
        }
        leaf targets {
          type string;
          description "The targets of the policy.
          Uuid of nodes and/or links separated by comma.";
        }
        leaf action {
          type string;
          description "The action of the policy";
        }

        description "Definition of policies to be
        utilized on the instantiation of the scenario.
        A policy is defined by a name, it type,
        the targets (nodes and/or links) to which it must
        be applied to, and the proper action that
        realizes the policy.";
      }

      description "Describes the deployment of all
      involved functional components mandatory for
      the execution of a benchmarking Test.";
    }
```

```
     grouping tool {
       leaf id {
         type uint32;
         description "The unique identifier of a tool.
         This information specifies how a tool can be
         identified in a list of probers/listeners of an
         Agent/Monitor.";
       }
       leaf instances {
         type uint32;
         description "The number of the tool instances that
         must be executed in parallel.";
       }
       leaf name {
         type string;
         description "The name of a tool.";
       }
       list parameters {
         key "input";
         leaf input {
             type string;
             description "The input key of a parameter";
         }
         leaf value {
             type string;
             description "The value of a parameter";
         }
         description "List of parameters for the execution
         of the tool. Each tool detains the proper set of running
         parameters that must be utilized to realize a benchmarking
         test.";
       }

       container sched {
         leaf from {
           type uint32;
           default 0;
           description "The initial time (in seconds)
           of the execution of the tool.";
         }

         leaf until {
           type uint32;
           description "The final/maximum time (in seconds)
           of the execution of the tool summed all its instances
           repeat, duration and interval parameters.";
         }
```

```
        leaf duration {
          type uint32;
          description "The total duration (in seconds) of the execution
          of each instance of the tool.";
        }

        leaf interval {
          type uint32;
          description "The interval (in seconds) to be awaited
          among each one of the instances of the
          execution of the tool.";
        }

        leaf repeat {
          type uint32;
          description "The number of times the tool must be executed.";
        }

        description "The scheduling parameters of a tool.
        Each Agent/Monitor must utilize the scheduling parameters
        to perform the execution of its tools (probers/listeners)
        accordingly.";
      }

      description "A tool to be used in a benchmarking test.
      A tool can be inferred as a prober or a listener.";
    }

    grouping component {
      leaf uuid {
        type string;
        description "A unique identifier";
      }
      leaf name {
        type string;
        description "The name of component";
      }

      description "A generic component.";
    }

    grouping agent {
      uses component;

      list probers {
        key "id";
        uses tool;
        description "Defines a list of the Prober(s)
```

```
        that must be used in a benchmarking test.";
      }
      description "An Agent defined by its uuid,
      name and the mandatory list of probers to be used
      by a benchmarking test.";
    }

    grouping monitor {
      uses component;

      list listeners {
        key "id";
        uses tool;
        description "Defines a list of the Listeners(s)
        that must used in a benchmarking test.";
      }
      description "A Monitor defined by its uuid,
      name and the mandatory list of probers to be used
      by a benchmarking test.";
    }

    grouping proceedings {
      list agents {
        key "uuid";
        uses agent;
        description "Defines a list containing the
        Agent(s) needed for a VNF-BD test.";
      }

      list monitors {
        key "uuid";
        uses monitor;
        description "Defines a list containing the
        Monitor(s) needed for a VNF-BD test.";
      }
      description "Information utilized by a Manager
      component to execute a benchmarking test.";
    }

    grouping vnf-bd {

      container experiments {
        leaf trials {
          type uint32;
          default 1;
          description "Number of trials.
          A trial is a single process or iteration
          to obtain VNF performance metrics from
```

```
          benchmarking the VNF-BD proceedings.";
        }
        leaf tests {
          type uint32;
          default 1;
          description "Number of tests.
          Each test defines unique structural
          and functional parameters (e.g., configurations,
          resource assignment) for benchmarked components
          to perform one or multiple Trials.
          Each Test must be executed following a
          particular scenario.";
        }
        description "Defines the number of trials and tests
        the VNF-BD must execute.";
      }

      container scenario {
        uses scenario;
        description "Scenarios defined by this VNF-BD.
        A scenario contains all information needed to describe
        the deployment of all involved functional components
        mandatory for the execution of a benchmarking Test.";
      }

      container proceedings {
        uses proceedings;
        description "Proceedings of VNF-BD.
        The proceedings are utilized by the Manager component
        to execute a benchmarking Test. It consists of
        agent(s)/monitor(s) settings, detailing their
        prober(s)/listener(s) specification and
        running parameters.";
      }

      description "A single VNF-BD.
      A VNF-BD contains all required definitions and
      requirements to deploy, configure, execute, and
      reproduce VNF benchmarking tests.";
    }

    uses vnf-bd;
  }
```

Figure 5

10.2. VNF Performance Profile

```
module vnf-pp {
  namespace "urn:ietf:params:xml:ns:yang:vnf-pp";
  prefix "vnf-pp";

  organization "IETF/BMWG";
  contact "Raphael Vicente Rosa <raphaelvrosa@gmail.com>,
  Manuel Peuster <peuster@mail.uni-paderborn.de>";

  description "Yang module for a VNF Performance Profile (VNF-PP).";

  revision "2019-10-15" {
      description "Reviewed VNF-PP structure -
      defines reports, snapshots, evaluations";
      reference "";
  }

  revision "2019-08-13" {
      description "V0.1: First release";
      reference "";
  }

  grouping tuple {
    description "A tuple used as key-value.";
    leaf key {
      type string;
      description "Tuple key.";
    }

    leaf value {
      type string;
      description "Tuple value.";
    }
  }


  grouping metric {
    leaf name {
      type string;
      description "The metric name";
    }

    leaf unit {
      type string;
      description "The unit of the metric value(s).";
    }
```

```
      leaf type {
        type string;
        mandatory true;
        description "The data type encoded in the value.
        It must refer to a known variable type, i.e.,
        string, float, uint, etc.";
      }

      choice value {
        case scalar {
          leaf scalar {
            type string;
            mandatory true;
            description "A single scalar value.";
          }
        }
        case vector {
          leaf-list vector {
            type string;
            min-elements 1;
            description "A list of scalar values";
          }
        }
        case series {
          list series {
            key "key";
            uses tuple;
            description "A list of key/values,
            e.g., a timeseries.";
          }
        }

        mandatory true;
        description "Value choice: scalar, vector, series.
        A metric can only contain a value with one of them.";
      }

      description "A metric that holds the recorded benchmarking
      results, can be a single value (scalar), a list of values
      (vector), or a list of key/value
      data (series), e.g., for timeseries.";

    }

    grouping evaluation {
        leaf id {
          type string;
          description "The evaluation
```

```
          unique identifier.";
        }

        leaf instance {
          type uint32;
          description "The unique identifier of the
          parallel instance of the prober/listener that
          was executed and created the evaluation.";
        }

        leaf repeat {
            type uint32;
            description "The unique identifier of the
            prober/listener repeatition instance
            was executed and created the evaluation.";
        }

        container source {

          leaf id {
            type string;
            description "The unique identifier of the source
            of the evaluation,
            i.e., the prober/listener unique identifier.";
          }

          leaf name {
            type string;
            description "The name of the source of the evaluation,
            i.e., the prober/listener name.";
          }

          leaf type {
            type string;
            description "The type of the source of the evaluation,
            i.e., one of prober or listener,  that was used to obtain
            it.";
          }

          leaf version {
            type string;
            description "The version of the tool interfacing
            the prober/listener that was used to obtain
            the evaluation.";
          }

          leaf call {
            type string;
```

```
            description "The full call of the tool realized by
            the source of the evaluation that performed
            the acquisiton of the metrics.";
          }

          description "The details regarding the
          source of the evaluation.";
        }

        container timestamp {

          leaf start {
            type string;
            description "Time (date, hour, minute, second)
            when the evaluation started";
          }

          leaf stop {
            type string;
            description "Time (date, hour, minute, second)
            when the evaluation stopped";
          }

          description "Timestamps of the procedures
          that realized the extraction of the evaluation.";
        }

      list metrics {
        key "name";
        uses metric;
        description "List of metrics obtained
        from a single evaluation.";
      }

      leaf error {
        type string;
        description "Error, if existent,
        when obtaining evaluation.";
      }

      description "The set of metrics and their source
      associated with a single Trial.";
    }


    grouping snapshot {
      leaf id {
        type string;
```

```
        description "The snapshot
        unique identifier.";
      }

      leaf trial {
        type uint32;
        description "The identifier of the trial
        when the snapshot was obtained.";
      }

      container origin {

        leaf id {
          type string;
          description "The unique identifier of the
          component of the origin of the snapshot,
          i.e., the agent or monitor unique identifier.";
        }

        leaf role {
          type string;
          description "The role of the component,
          origin of the snapshop, i.e.,
          one of agent or monitor.";
        }

        leaf host {
          type string;
          description "The hostname where the
          source of the snapshot was placed.";
        }

        description "The detailed origin of
        the snapshot.";

      }

      list evaluations {
        key "id";
        uses evaluation;
        description "The list of evaluations
        contained in a single snapshot Test.";
      }

      leaf timestamp {
        type string;
        description "Time (date, hour, minute, second)
        when the snapshot was created.";
```

```
      }

      leaf error {
        type string;
        description "Error, if existent,
        when obtaining the snapshot.";
      }

      description "The set of evaluations and their origin
      output of the execution of a single trial.";
    }

    grouping report {
      leaf id {
        type string;
        description "The report unique identifier.";
      }

      leaf test {
        type uint32;
        description "The identifier of the Test
        when the snapshots were obtained.";
      }

      list snapshots {
        key "id";
        uses snapshot;
        description "List of snapshots contained
        in a single report.";
      }

      leaf timestamp {
        type string;
        description "Time (date, hour, minute, second)
        when the report was created.";
      }

      leaf error {
        type string;
        description "Error, if existent,
        when obtaining the report.";
      }

      description "The set of snapshots output
      of a single Test.";
    }
```

```
    grouping header {
      leaf id {
        type string;
        description "Unique identifier of the VNF-PP.";
      }
      leaf name {
        type string;
        description "Name of the VNF-PP.";
      }
      leaf version {
        type string;
        description "Version of the VNF-PP.";
      }
      leaf description {
        type string;
        description "Description of the VNF-PP";
      }
      leaf timestamp {
        type string;
        description "Time (date, hour, minute, second)
        when the VNF-PP was created.";
      }

      description "The header content of a VNF-PP.";
    }

    grouping vnf-pp {

      uses header;

      list reports {
        key "id";
        uses report;
        description "List of the reports of a VNF-PP.";
      }

      description "A single VNF-PP.";
    }

    uses vnf-pp;
  }
```

Figure 6

10.3. VNF Benchmarking Report

```
module vnf-br {
  namespace "urn:ietf:params:xml:ns:yang:vnf-br";
  prefix "vnf-br";

  import vnf-bd {
        prefix "vnfbd";
        revision-date 2020-10-08;
  }

  import vnf-pp {
        prefix "vnfpp";
        revision-date 2020-10-08;
  }

  organization "IETF/BMWG";
  contact "Raphael Vicente Rosa <raphaelvrosa@gmail.com>,
  Manuel Peuster <peuster@mail.uni-paderborn.de>";
  description "Yang model for a VNF Benchmark Report (VNF-BR).";

revision "2020-09-09" {
      description "V0.2: Review the structure
      and the grouping/leaf descriptions.";
      reference "";
  }

  revision "2020-09-09" {
      description "V0.1: First release";
      reference "";
  }

  grouping variable {
    leaf name {
      type string;
      description "The name of the variable.";
    }
    leaf path {
      type string;
      description "The VNF-BD YANG path of the
      variable.";
    }
    leaf type {
      type string;
      description "The type of the
      variable values.";
    }
    leaf class {
```

```
        type string;
        description "The class of the
        variable (one of resource, stimulus,
        configuration).";
      }
      leaf-list values {
        type string;
        description "The list of values
        of the variable.";
      }
    }

    grouping output {
      leaf id {
        type string;
        description "The output unique identifier.";
      }
      list variables {
        key "name";
        leaf name { type string; }
        leaf value { type string; }
        description "The list of instance of varibles
        from VNF-BR:inputs utilized by a VNF-BD to
        generate a VNF-PP.";
      }

      container vnfbd {
        uses vnfbd:vnf-bd;
        description "The VNF-BD that was executed
        to generate a output.";
      }

      container vnfpp {
        uses vnfpp:vnf-pp;
        description "The output VNF-PP of the
        execution of a VNF-BD.";
      }
    }

    grouping vnf {
      leaf id {
        type string;
        description "The VNF unique identifier.";
      }
      leaf name {
        type string;
        description "The VNF name.";
      }
```

```
       leaf version {
         type string;
         description "The VNF version.";
       }
       leaf author {
         type string;
         description "The author of the VNF.";
       }
       leaf description {
         type string;
         description "The description of the VNF.";
       }
       description "The details of the VNF SUT.";
     }

     grouping header {
       leaf id {
         type string;
         description "The unique identifier of the VNF-BR ";
       }
       leaf name {
         type string;
         description "The name of the VNF-BR.";
       }
       leaf version {
         type string;
         description "The VNF-BR version.";
       }
       leaf author {
         type string;
         description "The VNF-BR author.";
       }
       leaf description {
         type string;
         description "The description of the VNF-BR.";
       }

       container vnf {
         uses vnf;
         description "The VNF-BR target SUT VNF.";
       }

       container environment {
         leaf name {
           type string;
           description "The evironment name";
         }
         leaf description {
```

```
            type string;
            description "A description
            of the environment";
          }
          leaf deploy {
            type boolean;
            description "Defines if (True) the environment enables
            the automated deployment by an orchestrator platform.";
          }
          container orchestrator {
            leaf name {
              type string;
              description "Name of the orchestrator
              platform.";
            }

            leaf type {
              type string;
              description "The type of the orchestrator
              platform.";
            }

            leaf description {
              type string;
              description "The description of the
              orchestrator platform.";
            }

            list parameters {
              key "input";
              leaf input {
                type string;
                description "The name of the parameter";
              }
              leaf value {
                type string;
                description "The value of the parameter";
              }

              description "List of orchestrator
              input parameters.";
            }

            description "The specification of the orchestration platform
            settings of a VNF-BR.";
          }

          description "The environment settings of a VNF-BR.";
```

```
      }

      description "Defines the content of a VNF-BR header.";
    }

    grouping vnf-br {
      description "Grouping for a single vnf-br.";

      uses header;

      container inputs {
        list variables {
          key "name";
          uses variable;
          description "The list of
          input variables.";
        }

        container vnfbd {
          uses vnfbd:vnf-bd;
          description "The input VNF-BD.";
        }

        description "The inputs needed to
        realize a VNF-BR.";
      }

      list outputs {
          key "id";
          uses output;
          description "The list of outputs
          of a VNF-BR.";
      }

      container timestamp {
        leaf start {
          type string;
          description "Time (date, hour, minute, second)
          of when the VNF-BR realization started";
        }

        leaf stop {
          type string;
          description "Time (date, hour, minute, second)
          of when the VNF-BR realization stopped";
        }

        description "Timestamps of the procedures that
```

```
        realized the realization of a VNF-BR.";
      }

      leaf error {
        type string;
        description "The VNF-BR error,
        if ocurred during its realization.";
      }
    }

    uses vnf-br;
  }
```

Figure 7

11. Acknowledgement

The authors would like to thank the support of Ericsson Research, Brazil. Parts of this work have received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. H2020-ICT-2016-2 761493 (5GTANGO: https://5gtango.eu).

12. References

12.1. Normative References

[ETS14a] ETSI, "Architectural Framework - ETSI GS NFV 002 V1.2.1", Dec 2014, <http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01-_60/gs_NFV002v010201p.pdf>.

[ETS14b] ETSI, "Terminology for Main Concepts in NFV - ETSI GS NFV 003 V1.2.1", Dec 2014, <http://www.etsi.org/deliver/etsi_gs/NFV/001_099-/003/01.02.01_60/gs_NFV003v010201p.pdf>.

[ETS14c] ETSI, "NFV Pre-deployment Testing - ETSI GS NFV TST001 V1.1.1", April 2016, <http://docbox.etsi.org/ISG/NFV/Open/DRAFTS/TST001_-_Pre-deployment_Validation/NFV-TST001v0015.zip>.

[ETS14d] ETSI, "Network Functions Virtualisation (NFV); Virtual Network Functions Architecture - ETSI GS NFV SWA001 V1.1.1", December 2014, <https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV-SWA%20001v1.1.1%20-%20GS%20-%20Virtual%20Network%20Function%20Architecture.pdf>.

[ETS14e] ETSI, "Report on CI/CD and Devops - ETSI GS NFV TST006 V0.0.9", April 2018, <https://docbox.etsi.org/isg/nfv/open/drafts/TST006_CICD_and_Devops_report>.

[ETS19f] ETSI, "Specification of Networking Benchmarks and Measurement Methods for NFVI - ETSI GS NFV-TST 009 V3.2.1", June 2019, <https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV-TST%20009v3.2.1%20-%20GS%20-%20NFVI_Benchmarks.pdf>.

[ETS19g] ETSI, "NFVI Compute and Network Metrics Specification - ETSI GS NFV-TST 008 V3.2.1", March 2019, <https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV-TST%20008v3.2.1%20-%20GS%20-%20NFVI%20Compute%20and%20Nwk%20Metrics%20-%20Spec.pdf>.

[RFC1242] S. Bradner, "Benchmarking Terminology for Network Interconnection Devices", July 1991, <https://www.rfc-editor.org/info/rfc1242>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC8172] A. Morton, "Considerations for Benchmarking Virtual Network Functions and Their Infrastructure", July 2017, <https://www.rfc-editor.org/info/rfc8172>.

[RFC8204] M. Tahhan, B. O'Mahony, A. Morton, "Benchmarking Virtual Switches in the Open Platform for NFV (OPNFV)", September 2017, <https://www.rfc-editor.org/info/rfc8204>.

12.2. Informative References

[gym] "Gym Framework Source Code", <https://github.com/intrig-unicamp/gym>.

[Peu-a] M. Peuster, H. Karl, "Understand Your Chains: Towards Performance Profile-based Network Service Management", Fifth European Workshop on Software Defined Networks (EWSDN) , 2016, <http://ieeexplore.ieee.org/document/7956044/>.

[Peu-b] M. Peuster, H. Karl, "Profile Your Chains, Not Functions: Automated Network Service Profiling in DevOps Environments", IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN) , 2017, <http://ieeexplore.ieee.org/document/8169826/>.

[Peu-c] M. Peuster, H. Karl, "Understand your chains and keep your deadlines: Introducing time-constrained profiling for NFV", IEEE/IFIP 14th International Conference on Network and Service Management (CNSM) , 2018, <https://ris.uni-paderborn.de/record/6016>.

[Peu-d] M. Peuster and S. Schneider and H. Karl, "The Softwarised Network Data Zoo", IEEE/IFIP 15th International Conference on Network and Service Management (CNSM) , 2019, <https://sndzoo.github.io/>.

[RFC3688] Mealling, M., "The IETF XML Registry", January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[Rosa-a] R. V. Rosa, C. E. Rothenberg, R. Szabo, "VBaaS: VNF Benchmark-as-a-Service", Fourth European Workshop on Software Defined Networks , Sept 2015, <http://ieeexplore.ieee.org/document/7313620>.

[Rosa-b] R. Rosa, C. Bertoldo, C. Rothenberg, "Take your VNF to the Gym: A Testing Framework for Automated NFV Performance Benchmarking", IEEE Communications Magazine Testing Series , Sept 2017, <http://ieeexplore.ieee.org/document/8030496>.

[Rosa-c] R. V. Rosa, C. E. Rothenberg, "Taking Open vSwitch to the Gym: An Automated Benchmarking Approach", IV Workshop pre-IETF/IRTF, CSBC Brazil, July 2017, <https://intrig.dca.fee.unicamp.br/wp-content/plugins/papercite/pdf/rosa2017taking.pdf>.

[tango] "5GTANGO: Development and validation platform for global industry-specific network services and apps", <https://5gtango.eu>.

[tng-bench] "5GTANGO VNF/NS Benchmarking Framework", <https://github.com/sonata-nfv/tng-sdk-benchmark>.

Authors' Addresses

Raphael Vicente Rosa (editor)
University of Campinas
Av. Albert Einstein, 400
Campinas, Sao Paulo 13083-852
Brazil

Email: rvrosa@dca.fee.unicamp.br
URI: https://intrig.dca.fee.unicamp.br/raphaelvrosa/

Christian Esteve Rothenberg
University of Campinas
Av. Albert Einstein, 400
Campinas, Sao Paulo 13083-852
Brazil

Email: chesteve@dca.fee.unicamp.br
URI: http://www.dca.fee.unicamp.br/~chesteve/

Manuel Peuster
Paderborn University
Warburgerstr. 100
Paderborn 33098
Germany

Email: manuel.peuster@upb.de
URI: https://peuster.de

Holger Karl
Paderborn University
Warburgerstr. 100
Paderborn 33098
Germany

Email: holger.karl@upb.de
URI: https://cs.uni-paderborn.de/cn/