Taking Open vSwitch to the Gym: An Automated Benchmarking Approach

Raphael Vicente Rosa[1], **Christian Esteve Rothenberg**[1]

[1]School of Electrical and Computer Engineering (FEEC)
University of Campinas (UNICAMP)
Campinas – SP – Brazil

{raphaelvrosa,chesteve}@dca.fee.unicamp.br

Abstract. *Performance benchmarking in Network Function Virtualization (NFV) pose challenging issues due to all moving parts of virtualized infrastructures potentially affecting the packet processing performance of Virtualized Network Functions (VNFs). Despite the advances in best-of-breed network virtualization technologies, the dependencies on the underlying allocated hardware resources, their characteristics and customized configurations, result in benchmarking hazards that call for innovative and standardized testing methodologies towards adequate VNF performance profiling. To this end, we designed and prototyped Gym, a testing framework for automated NFV performance benchmarking we experimentally validated on Open vSwitch as a target VNF. The design principles and implementation of Gym demonstrate an useful apparatus to assist standards organizations formalizing VNF testing methodologies.*

1. Introduction

Our initial work on VNF Benchmarking as a Service (VBaaS) [Rosa et al. 2015a] introduced the problem statement of VNF benchmarking based on 'trust-but-verify' principles towards standardized performance testing to evaluate candidate platforms and locations to host the (chains of) VNFs with respect to target Key Performance Indicators (KPIs). Presented in Internet Research Task Force (IRTF) NFV Research Group (NFVRG), [Rosa et al. 2015b] brought ideas towards the problem statement and initial handling proposal for VBaaS. At that time, attention was given to VNF performance profiles needed for Network Function Virtualization Orchestrator (NFVO) embedding algorithms as well as parameters in support of business decisions including resource optimization objectives and Service Level Agreement (SLA) related aspects of NFV contracts. In a more focused approached, our efforts moved to initiating a discussion towards "VNF Benchmarking Methodology" [Rosa et al. 2016] inside the Benchmarking Methodology Working Group (BMWG). However, there was no further traction due to the lack of solid results that could assist the common relationship of Internet Engineering Task Force (IETF) with rough consensus and running code.

Ongoing work at IETF BMWG under the umbrella of "Considerations for Benchmarking Virtual Network Functions and Their Infrastructure" [Morton 2016], brings important guidelines and initial directions for developing standardized VNF benchmarking methodologies. Alongside, "Benchmarking Virtual Switches in OPNFV" [Tahhan et al. 2016] initiates the discussion on a methodology to benchmark software switches based on experiences in OPNFV VSperf project. Simi-

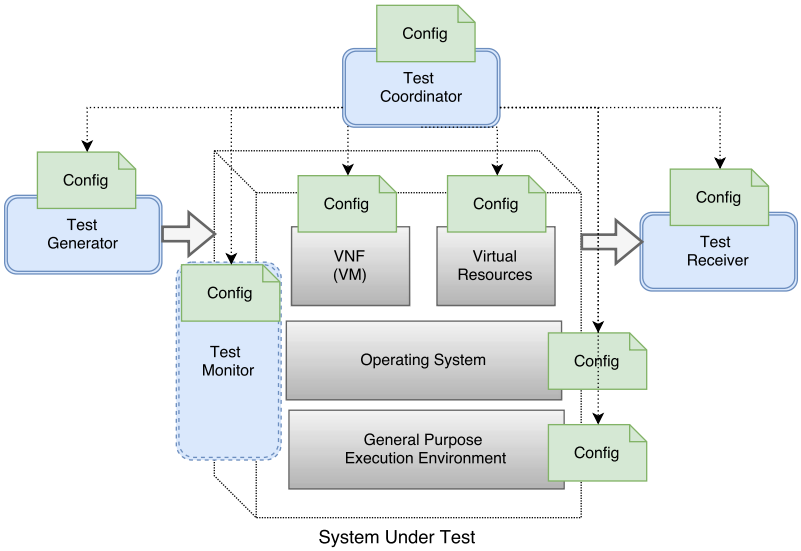


Figure 1. Big picture of VNF Testing

larly, the European Telecommunications Standards Institute (ETSI) Industry Specification Group (ISG) NFV Testing Group Specification Report on Pre-deployment Testing [ETSI GS NFV-TST 2016] details requirements and recommendations for validating VNFs and NFV Infrastructure (NFVI), with especial attention on the separation of control and data plane characteristics.

Revisiting our initial ideas on VBaaS [Rosa et al. 2015a] towards moving forward on the IETF/IRTF standardization threads by means of running code, in this paper, we present Gym as a testing (see Fig. 1) framework implementation based on a minimum set of standardized interfaces allowing user-defined tests along a catalog of reusable VNF testing reports and procedures with wide- and well-defined system configurations, workload parametrization (linking to specific traffic generation tools and their configuration), KPI computation, along all data expected from standardized benchmark methodologies. As argued in [Raumer et al. 2016], methodologies of benchmarking to interconnect devices can be revisited by new perspectives of measurements (e.g., latency long tail behavior) and new tool-sets (e.g., MoonGen [Emmerich et al. 2015]).

Already in production networking use cases (e.g., Google B4 [Jain et al. 2013]), software switches demand a particular set of data plane design and configurations in order to attain desired packet processing performance. In a seminal work [Pfaff et al. 2015], Open vSwitch is dissected highlighting main concerns in expressiveness of OpenFlow versus performance, and how some design and implementation decisions were taken to solve such issue. Another recent work [Molnár et al. 2016] shows the importance of data plane specialization in attaining high performance for OpenFlow-like pipelines in a switch built over Intel Data Plane Development Kit (DPDK). Motivated by the related work and relevance of programmable software switches (as a cornerstone VNF), we illustrate Gym (dissected in Sec. 2) throughout our benchmarking experiments with OpenvSwitch (OVS) (see Sec. 3). We present a critical analysis of our findings and design principles in Sec. 4 and discuss related work in Sec. 5. Finally, we conclude the paper in Sec. 6 with pinpoints and future work.

2. Gym: An Automated Benchmarking Framework

Our approach towards the Gym testing framework is based on providing proper abstractions and instantiation manners to validate, benchmark and dimension VNFs [ETSI GS NFV-TST 2016], leveraging as much as possible the existing literature (e.g., IETF BMWG) and practical work (e.g., open source components), and altogether calling for a community approach following open data and open source practices.

During the design and development process of the former VBaaS [Rosa et al. 2015a], new abstractions were upgraded and implemented, giving birth to sets of generic and modular VNFs testing components. The overall implemented framework was then baptized as Gym. Among its main characteristics, we highlight: (*i*) modular architecture with stand-alone programmable components; (*ii*) simple messaging system among them following generic Remote Procedure Call (RPC) guidelines; (*iii*) extensible set of tools and associated metrics; (*iv*) programmability of tests through dynamic compositions of modules; and (*v*) corresponding flexible methods for output processing and visualization of tests results.

As a software-centric framework for NFV, Gym seeks to introduce new opportunities to different actors. As shown in Fig. 1, VNF developers can rely on the framework to develop tests for VNFs performance profiling aligned with agile Continuous Integration of DevOps methodologies to speed their time-to-market. Service Providers might enhance offered services Quality of Service (QoS) with tested-deployed scenarios (e.g., varying workloads in multiple sites), containing transparent sets of required VNF metrics. Infrastructure/Cloud Providers, via VNF testing in their execution environments, could increase reliability with compliance methodologies (e.g., energy consumption).

2.1. Conceptual Ideas and Guiding Principles

Design for modularity is one of the main guiding principles of Gym to allow independent software components being orchestrated on-demand to fulfill the testing objectives and provide unconstrained means for extensibility. Considering the VNFs complex set of requirements and capabilities, the framework gives a high degree of freedom to users to compose sets of tools and evaluation models through simple description formats to continuously develop and integrate VNFs testing methodologies. Gym design principles, enunciated below, come into discussion further with the evaluation of a VNF benchmarking use case. When such principles are adopted in designing and building a testing framework, we believe experimentation methodologies can be written in simple ways for testing VNFs (e.g., dimensioning or benchmarking).

1. **Comparability:** output of tests shall be simple to understand and process, in a human-readable format, coherent and easily reusable (e.g. inputs for analytic applications).
2. **Repeatability:** testing setup must be comprehensive defined through a handful/flexible design model, which can be interpreted and executed by the testing platform repeatedly, allowing customization.
3. **Interoperability:** tests should be able to be ported to different environments using lightweight technologies.
4. **Configurability:** open interfaces and extensible messaging models between components to provide flexibility when composing test descriptions and configurations.

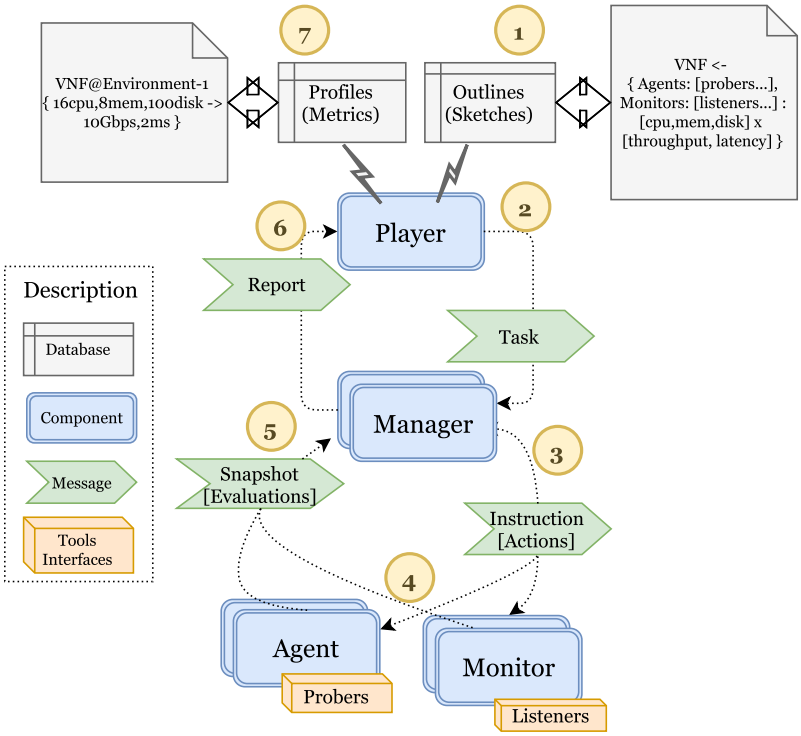


Figure 2. Gym Architecture

2.2. Architecture

The main components of Gym are illustrated in Fig. 2.

Agent. Towards extensible (e.g., plug-and-play) tools' interfaces (e.g., iperf, ping), named probers, executes active probes to collect network and host-specific metrics. While a single Agent is capable of performing localized tests (e.g., CPU and disk I/O benchmarks), the interaction among distributed agents enables the collection of end-to-end network metrics (e.g., frame loss rate, latency). Possibly, one end can be a VNF itself where, for example, one-way throughput is evaluated. Agent's APIs are open and modular for flexible extensions. It receives from Manager procedure calls with *instructions* containing a set of actions to properly run probers, parse their results, and send back *snapshots* containing the set of evaluations of those probers' actions.

Monitor. Designed for internal and external instrumentation of VNFs and their execution environment, aims to perform passive monitoring via tools' interfaces (e.g., top, tcpdump), named listeners, for metrics collection according to Agents' workloads. For instance, to monitor vCPU utilization of a VNF execution environment when Agents probe traffic is sent (e.g., with particular packet length). Different from Agents' active approach, seen as generic VNF probers, Monitors observe particular properties according to capabilities offered by VNFs and their respective execution environment. Similarly to Agents, Monitors interact with Manager to receive *instructions* and then to reply *snapshots*.

Manager. Responsible for (a) keeping a coherent state of controlled components (e.g., Agents and Monitors) and their features; (b) interacting with Player to receive *task*s and decompose them into a coherent set of *instruction*s (containing actions) for; (c) the synchronous coordination of Agents and Monitors' activities; and (d) processing *snapshots*

with their proper aggregation into *reports* back to Player.

Player. As the main entry-point for users, it defines interfaces with different abstractions: (i) metrics correspondent to probers or listeners and their associated properties and requirements; (ii) and VNF testing *outlines* containing one or more metrics with their associated parameters. Player receives inputs as *outline*s and decomposes them in a set of *task*s necessary to extract their associated metrics. Such *task*s are selected to be sent to Manager and then to receive *report*s, which can be properly parsed and saved in databases. An interface is provided for graphic visualization purposes of the extracted metrics.

Two relevant terms deserve further explanation:
· *Outline:* specifies how to test a VNF that may be specific to it or applicable to several VNF types. It includes structural (e.g., Agents/Monitors) and functional *sketches* with variable parameters (e.g., probers/listeners properties), used as inputs by Gym to perform VNFs tests.
· *Profile:* is composed by the outputs of an Outline execution, and represents a mapping between virtualized resources (e.g., vCPU, memory) and performance (e.g., throughput, latency between in/out or ports) at a given environment. It abstracts a VNF allocation with desired resources to deliver a given (predictable/measured) performance quality.

2.3. Messaging System and Workflow

Gym core components communicate through REpresentational State Transfer (REST) Application Programming Interface (API) using generic RPC calls with custom JSON message formats. We now describe request-reply message exchanges within the pairwise component interactions as represented in the numbered (1 to 7) circles of Fig. 2.

1. The first step consists of a user defining the composition of the VNF testing *Outline* through *Sketches* as the structural and functional requirements to express target performance metrics to generate a VNF *Profile*.
2. The Player processes the parametrized *Outline* considering the features offered by its associated Managers. The job's output is a workflow of *tasks*, in sequence or parallel, submitted to a selected Manager that satisfies (controls set of Agents/Monitors matching) the *Outline* requirements. Based on input variables, an *Outline* can be decomposed in sets of tasks, high-level *probers/listeners* parameters and inherited *Outline* input variables.
3. The Manager decomposes *tasks* in a coherent sequence of *instructions* to be sent to Agents and/or Monitors. Inside each *instruction*, sets of *actions* define execution procedures of *probers/listeners* along the respective parameters. Sequential or parallel *tasks* might have properties to be decomposed in different sets of *instructions*, for instance, when sampling cycles might define the execution of *instructions* repeatedly.
4. By interpreting *actions* into *probers/listeners* execution, Agents and Monitors perform active and passive measurements to output metrics via pluggable tools. For instance, a VNF developer can freely create customized probers and listeners to interface her tests and extract particular metrics. Such tools' interfaces are automatically discovered by Agents/Monitors and exposed as "available" to Manager and Player with their proper execution parameters and output properties.
5. After extracting required metrics, on the way back, sets of *evaluations* (*actions* parsed outputs) integrate *snapshots*, which are sent from Agents/Monitors to the

Manager. All sets of *snapshots* dependent on a specific *task* are received from Agents/Monitors whom *instructions* were sent to. *Evaluations* contain timestamps and identification of the originating probers and listeners, whereas *snapshots* receive Agents/Monitors unique identification and their environment hostname.

6. After processing all the *instructions'* related tree of *snapshots*, the Manager composes a *report*, as a reply to each *task* requested by the Player. The Manager can sample *snapshots* in a diverse set of programmable methods. For instance, a *task* may require cycles of repetition, so the correspondent *snapshots* can be parsed and aggregated in a report through statistic operations (e.g., mean, deviation).
7. Finally, the Player processes the *report* following the *profile* metrics definition, as established initially during the *outline* decomposition. While the *profile* contains filtered evaluation metrics and parameters, the individual *snapshots* can be aggregated/sampled into *reports*. Results can be exported in different file formats (e.g., csv, json, yaml) or saved into databases for further analysis and visualization. For instance, in our current Gym instantiation we integrate the well-known open source Elasticsearch database and the Kibana visualization platform —tools providing high flexibility in querying, filtering and creation of different visual representations of the extracted *profiles*.

3. Benchmarking Open vSwitch

Herein we present our experiments with Gym when benchmarking Open vSwitch, which was our benchmarking System Under Test (SUT) choice, because of being widely used over many Linux distributions, virtualization environments (e.g., Xen, KVM, Docker) and platforms (e.g., OpenStack, OpenNebula). From openvswitch.org, "Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (e.g. NetFlow, sFlow, SPAN, RSPAN, CLI, LACP, 802.1ag)."

3.1. Scenario

Our testing environment (see Fig. 3) was designed as a initial prototype scenario to benchmark OVS. It consists of three Virtual Machines (VMs) in the same host. The first one, named *Source*,contains an Agent triggering probers to generate traffic as benchmarking stimulus in the second VM. There, OVS is the SUT attached to two interfaces interconnecting two isolated networks, between the first and third VMs. Alongside OVS, there is a Monitor component executing the listeners that measure resources consumption in the VM. In *Destination*, third VM, an Agent is placed, providing probers that might be used to benchmark OVS. In the host, Player and Manager are executed, communicating with all the other Gym components via an isolated management network.

Each VM has 1 core of CPU, 2048 MB of memory and 15 GB of disk allocated. Ubuntu Server version 16.02TLS is the default operating system in all VMs, which also only contain the necessary packages to run Gym components, with the exception of *SUT*, where OVS version 2.5.0 is additionally installed. Proper configurations of routing and ARP tables were done in *Source* and *Destination* to be pairwise reachable, and flow entries were set in OVS to forward traffic in forth and back directions between the VMs based on their attached ports, source and destination IP and MAC addresses.

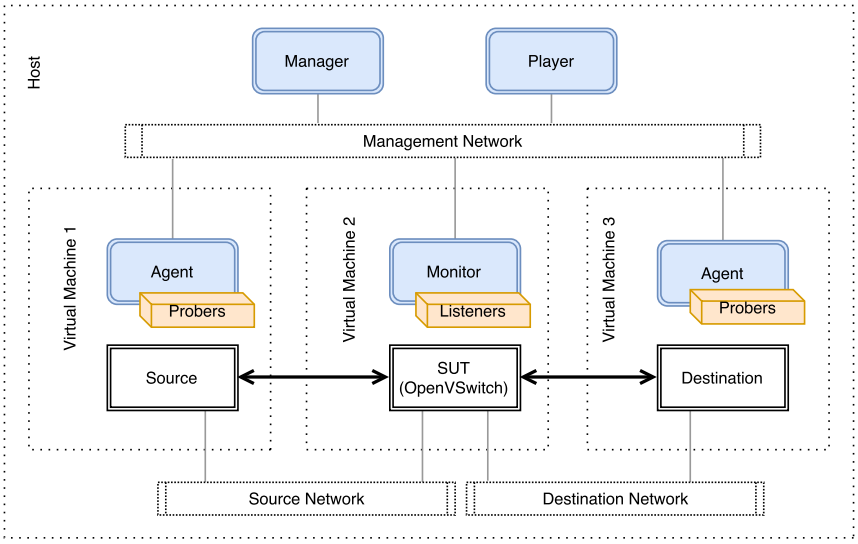


Figure 3. Open vSwitch Benchmarking Scenario

3.2. Methodology

Initially, a deep analysis on the literature gave us glimpses of OVS benchmarking, meaning the approach we used to extend Gym and create a methodology for the experiments. Being widely used and in high level of maturity, *pktgen* was our target tool to stimulate OVS. We adopted a well-known interface in python language, known as *psutil* library, to monitor OVS resources consumption, as an internal instrumentation method. For Gym, it meant creating a new prober for *pktgen* and a new listener for *psutil*. Their parameters and output metrics are presented in Table 1.

Table 1. Gym extensions: metrics and parameters

	prober pktgen	**listener process**
Parameters	{{ device }} {{ duration }} {{ frame_size }} {{ source_ip }} {{ source_mac }} {{ destination_ip }} {{ destination_mac }}	{{ pid_name }} {{ interval }} {{ duration }}
Metrics	packets_sent packets_per_second bits_per_second megabits_per_second errors	cpu disk memory

Along with such extensions, main design principles of Gym come to the scene, where *Sketches* were composed to define the OVS testing *Outline*, presented in Fig. 4 in YAML[1] format. In details, a *sketch* contains unique a identifier and name, inputs that will be used to fill the required parameters, requirements defining Gym components and their respective probers/listeners type (unique identification in Gym tools catalog). In addition, taking a *uuid*, internally unique identifier in the *sketch*, parameters can be set by inputs that will represent the prober/listener or host configurations. For instance, in

[1]http://www.yaml.org/

listener process *sketch*, the *host* input, if defined, can specify the hostname in which the listener process is desired to be executed.

Open vSwitch composed *Outline* has a different structure. Besides, identifier, name and inputs, it also contains: *vars* that might be used internally; *groups* that can compose groupings of hosts to be used as inputs in sketches (e.g., when multiple agents in different hosts are required to run the same prober(s)); *sketches* containing the reference to filename of the *sketch* and its respective input fields; and cycles defining temporal parameters to characterize the decomposition of the *outline* into *tasks*.

Sketch:
listener process

```
id: 18
name: 'listener-process'

inputs:
  pid: ''
  name: ''
  duration: 30
  interval: 1
  host: ''

requirements:
  monitors:
    - uuid: 1
      listeners:
        - uuid: 100
          type: 18

parameters:
  - uuid: 100
    name: {{ name }}
    pid: {{ pid }}
    duration: {{ duration }}
    interval: {{ interval }}

  - uuid: 1
    host: {{ host or '' }}
```

Sketch:
prober pktgen

```
id: 5
name: 'bandwidth-pktgen'

inputs:
  device: ''
  duration: ''
  frame_size: ''
  src_ip: ''
  src_mac: ''
  dst_ip: ''
  dst_mac: ''
  rate_bps: ''
  rate_pps: ''
  threads: ''
  pkt_clones: ''
  burst: ''
  host: ''

requirements:
  agents:
    - uuid: 1
      probers:
        - uuid: 500
          type: 5

parameters:
  - uuid: 500
    device: {{ device }}
    duration: {{ duration }}
    frame_size: {{ frame_size }}
    src_ip: {{ src_ip }}
    src_mac:  {{ src_mac }}
    dst_ip:  {{ dst_ip }}
    dst_mac:  {{ dst_mac }}
    rate_bps: {{ rate_bps }}
    rate_pps:  {{ rate_pps }}
    threads:  {{ threads }}
    pkt_clones:  {{ pkt_clones }}
    burst: {{ burst }}

  - uuid: 1
    host: {{ host }}
```

Outline:
Open vSwitch

```
id: 3
name: 'OVS-benchmarking'

inputs:
  host_src: 'src'
  mon_process_name: 'ovs-vswitchd'
  host_mon: ''
  device: ''
  duration: 30
  frame_size: 64
  src_ip: ''
  src_mac: ''
  dst_ip: ''
  dst_mac: ''
  interval: 1

vars:
  var1: '1000'

groups: null

sketches:
  - source: 'process.yaml'
    groupings: null
    inputs:
      name: {{ mon_process_name }}
      duration: {{ duration }}
      interval: {{ interval }}
      host: {{ host_mon }}

  - source: 'bandwidthv4.yaml'
    groupings: null
    inputs:
      device: {{ device }}
      duration: {{ duration }}
      frame_size: {{ frame_size }}
      src_ip: {{ src_ip }}
      src_mac:  {{ src_mac }}
      dst_ip:  {{ dst_ip }}
      dst_mac:  {{ dst_mac }}
      host: {{ host_src }}

cycles:
  when: null
  repeat: 0
  every: 60
```

Figure 4. Open vSwitch Sketches and Outline

In details, a *sketch* can contain reference to a set of requirements (e.g., multiple Agents and/or Monitors and respective probers/listeners), and *outline* can be composed by any structure of *sketches* and their respective inputs. So far, inputs are not defined as mandatory or optional, but full freedom is given on such specification to developers compose *sketches* and *outlines* with their own custom needs. In *outlines*, groups can be composed by a list of hosts that, when used as input parameter in a *sketch*, will define that

every required host will attend the desired *sketch* requirements equally and, therefore, execute in parallel the same set of probers/listeners.

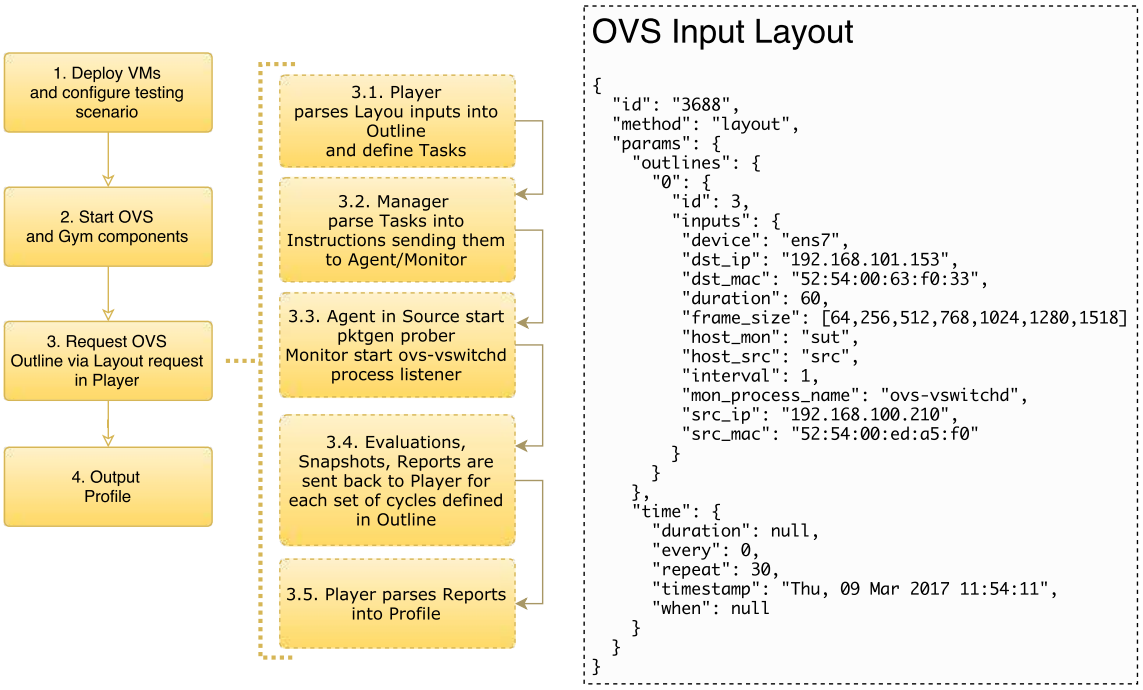


Figure 5. Open vSwitch Benchmarking Methodology

Figure 5 present the methodology created to execute the prototype benchmarking tests alongside the *Layout* (JSON format) that Gym receives in Player to trigger the execution of the tests with the required input parameters. It is important to note the identifier of the *outline* with its input parameters are provided in the *Layout*. Note the field *frame_size* where a list of items was defined and in *cycles* the parameter *repeat* equals 30. With such inputs, Player will identify the required *Outline*, set its inputs, checking *frame_size* as being a list, and create for each of the items in that list, a *task* to be sent to Manager. As *repeat* input is set to 30, every task will be sent thirty times.

Finished the execution of the experiments triggered by the input *Layout*, results are visualized using the Kibana[2] interface, as Player saves all output Profiles in Elasticsearch database[3]. Following the syntax query provided by such tools, multiple combinations of metrics can be visualized in a variety of graphic formats (e.g., table, pie chart, areas/lines, time series, etc). In addition, queries to the database can be done using Elasticsearch syntax to retrieve *Profile* results and create custom graphics with any other set of tools (e.g., export to CSV).

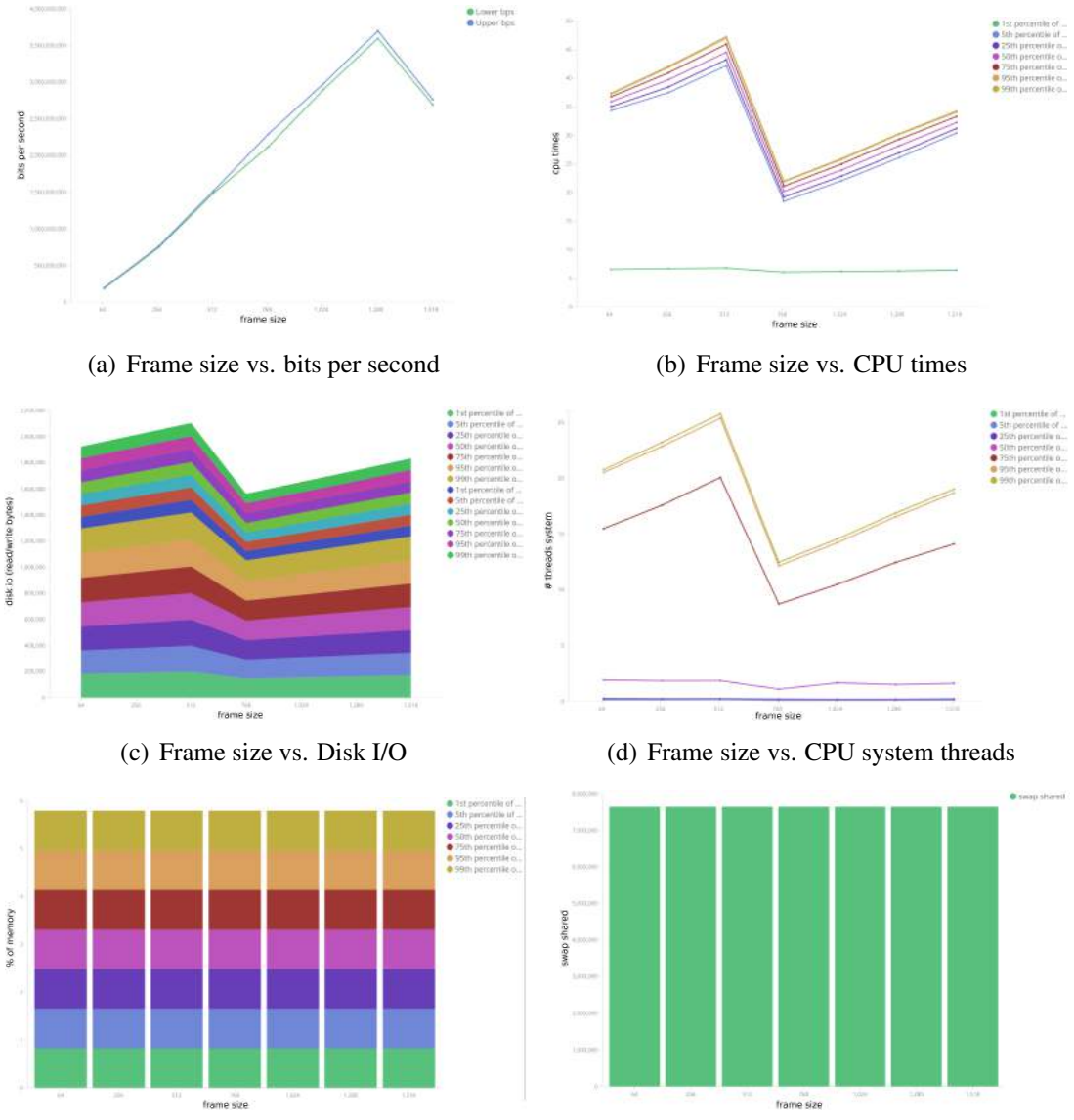
3.3. Experimental Analysis

Figure 6 presents filtered results in a series of graphics. According to *pktgen* prober input parameters, only duration, frame size, IP and MAC (source and destination) addresses were provided for UDP packets being sent for 60 seconds at the maximum rate allowed.

[2]https://www.elastic.co/products/kibana
[3]https://www.elastic.co/products/elasticsearch

Figure 6(a) shows a correlation between the increase in the frame size and bits per second (upper and lower standard deviation) registered by *pktgen* output metrics. In the resource consumption perspective, *ovs-vswitchd* was monitored while *pktgen* generated traffic in OVS SUT. Fig. 6(b), gives an indication of CPU consumption by *ovs-vswitchd*, showing how much CPU times were consumed in different percentiles by each different stimulated frame size traffic.

Figure 6(c) shows disk Input/Output read and write of bytes in different percentiles according to the stimulated *pktgen* traffic frame sizes. Fig. 6(d) similarly shows how different frame sizes trigger the usage of a different amount of threads used by the process *ovs-vswitchd*. And in the last two Figures, 6(e) and 6(f), visually it is possible to notice that different frame sizes did not incur in alterations in the levels of percentage of memory and swap by *ovs-vswitchd* process.



(a) Frame size vs. bits per second

(b) Frame size vs. CPU times

(c) Frame size vs. Disk I/O

(d) Frame size vs. CPU system threads

(e) Frame size vs. Memory

(f) Frame size vs. Swap

Figure 6. Results: Open vSwitch Output Profile

4. Discussion

As an initial scenario of benchmarking testing, the prototype under evaluation presents measurements exhibiting expected behaviors. An increased level of bandwidth was seen as frame sizes were increased, not affecting the memory or swap consumption by the main OVS process, however leading similar patterns of behavior in CPU and disk I/O consumption, possibly triggered by internal OVS caching behaviors. A limitation of bits per second was seen in frames with 1518 bytes, however further analysis is needed to explain if OVS could not handle such traffic workload or the *Source* VM was the bottleneck due to the allocated resources hampering the *pktgen* prober to operate at full capacity.

4.1. Gym Design Principles Assessment

While our experimental methodology was carried on a limited set of VMs and a small combination of Gym components, the experiences benchmarking OVS give us some inputs for an initial critical analysis on Gym design principles.

* *Comparability:* all presented results were extracted from the obtained OVS Profile by Gym Player component. The presented data could be exported in various file formats with multiple combinations of metrics association. In our case, it was graphically shown in Fig. 6. Therefore, any logical data parsing method can be formalized based on the flexibility of Gym saving Profiles in Elasticsearch database.
* *Configurability:* on designing the *Sketches* and *Outline* to benchmark the OVS, total freedom of choice to program customized experimentation methodologies was possible by deploying diverse requirements and input parameters associated with multiple Gym components. Full autonomy is given here to developers compose their own custom sets of test recipes in *sketches* and *outlines*.
* *Repeatability:* using Gym and the same testing scenario, the *sketches* and *outline* developed for our benchmarking tests can be used repeatedly to replicate the presented results. Besides, with the modification of the running scenario our running experiments can be easily replicated (e.g., in case OVS were deployed in container by Kubernetes orchestrator).
* *Interoperability:* all Gym components can be deployed in heterogeneous environments (i.e., main requirements sit on support of the Python programming language). In addition, multiple new technologies can be attached to Gym as southbound interfaces in probers/listeners, and northbound APIs over Player.

4.2. On Automating VNF Benchmark

After identifying Gym design principles in sustaining main arguments towards its usage in benchmarking VNFs, below we introduce some topics and analysis regards bringing automation mechanisms to benchmarking.

Scenario: network function as well source and destination traffic components might be virtualized or not. Commonly seen bare-metal black-box based benchmarking methodologies distinguish from virtualized scenarios where, nowadays orchestration platforms (e.g., OpenStack, Juju, Kubernetes) might be used to deploy the experimental scenario. Our beliefs with Gym follow such separation between scenario and experiments configuration/execution in order to create test modularity, flexibility and agility – practices commonly seen and needed in (DevOps) continuous development and integration of VNFs.

Gym, comes as a standalone framework useful in orchestrated scenarios (virtualized or not) to compose new probers/listeners, custom *sketches*, *outlines*, and output test Profiles.

Configuration: tests might need particular set of custom configurations in the underlying hardware capabilities and in the VNF itself (e.g., as made before benchmarking OVS). Configuration scripts might enhance the capabilities offered by different scenarios (e.g., use huge pages of memory) and procedures taken to optimize hardware and software components (e.g., custom DPDK parameters) for high performance measurements. Besides, even during series of tests, routine procedures might be needed to adjust SUT and execution environment for continuous automated benchmarks (e.g., clear memory caches, reinstall flow entries, restart monitoring processes). Gym currently lacks such features, however our development views aim integrating current DevOps frameworks (e.g., Puppet[4], Chef[5] and Salt[6]) for automated configuration of tests.

Execution: Gym poses APIs and information models open for extension, and its flexible messaging model based on JSON-RPC allows full customization of the entire framework. In discussed topics composing experimentation methodologies (i.e., comparability, configurability, repeatability, interoperability), Gym allows high degree of testing composition and expressiveness via *sketches* and *outlines*, at same time that provides interfaces (probers/listeners) to any sort of benchmarking tools (virtualized or not). Taking those features in consideration, present Gym automation can allow the extraction of new types of testing strategies (e.g., measuring system failure/errors, VNF elasticity, noisy behavior, etc [Morton 2016]). Moreover, Gym is currently receiving new management and operational features (e.g., debugging, controllability, reconfigurability, etc).

Output: Among the main features required in testing frameworks, plotting visual interfaces are extremely necessary. Graphically, behavior patterns and outliers can be easily identified. Visually speaking, many graphic libraries can be attached to Gym, as it allows VNF Profiles to be saved in different file formats, and always be added to Elasticsearch database. Timestamps are defined in each JSON-RPC reply message, meaning all of them (evaluations, snapshots, reports and profiles) can be granularly plotted in time series. Current Kibana integration allows a high degree in creating visualization graphics and dashboards of VNF Profiles. In addition, specialized analytics methods (e.g. clustering) are being introduced in Gym extensions, specially focused on creating new automated visual possibilities of Profiles to better examine VNFs performance.

5. Related Work

Existing open source projects sprint common abstractions for benchmarking VNFs and their underlying infrastructure. In OPNFV, highlights go for three of them. Yardstick[7] targets infrastructure compliance when running VNF applications. QTIP[8] approaches provides definitions towards platform performance benchmarking. Whereas Bottlenecks[9] proposes a framework to execute automatic methods of benchmarks to validate VNFs deployment during staging.

[4]https://puppet.com/
[5]https://www.chef.io/chef/
[6]https://saltstack.com/
[7]https://wiki.opnfv.org/yardstick
[8]https://wiki.opnfv.org/display/qtip/Platform+Performance+Benchmarking
[9]https://wiki.opnfv.org/display/bottlenecks

Closest to our efforts, OPNFV VSperf project[10], with experiences described in [Tahhan et al. 2016], presents "an automated test-framework and comprehensive test suite based on industry standards for measuring data-plane performance of Telco NFV switching technologies as well as physical and virtual network interfaces (NFVI)". VSperf gives the freedom to test-customizations (e.g., software components, load generators) being suitable for different switching technologies in a telco NFV environment.

Recently, Canonical released a set of solutions to use Juju, its service modeling and execution platform, to create the means to leverage their automated cloud deployment services with benchmarking instruction and action models[11]. Therefore, benchmarks can be composable against a variety of hardware configuration, architecture, and cloud applications. In addition, an independent but related effort is ToDD[12], which walks in the direction of an on-demand distributed abd highly extensible framework for distributed capacity and connectivity testing.

6. Conclusion and Future Work

In consequence of NFV evolving technologies to design suitable carrier-grade VNFs execution, and while agility is needed for the recognition of their adequate placement capabilities, ways of identifying correlations between packet processing performance and resources allocation are open research fields advocated in this paper. Following this path, we introduce Gym, a testing framework for automated NFV performance benchmarking. Based on the current motivations of developing VNF benchmarking methodologies in standards organizations (e.g. IETF), we pose Gym in face of Open vSwitch. Through Gym design principles assessments, we highlight the freedom of modularity in composing benchmarking tests for software switches and retrieving measurements for proper analysis and graphical results. Presented experiments show a fair evaluation of OVS with current state-of-the-art software switch benchmarking literature. Additionally, reviewing Gym design principles and automation topics, critical analysis of the efforts so far prototyped in Gym indicate it as a prominent VNF benchmarking framework.

We believe the concepts introduced in this paper might help on the understanding developments of standard VNF benchmarking methodologies in a broad sense, taking into account latest efforts made by ETSI and IETF/IRTF. Fully intended to free Gym as an open source project, our future goals sit on developments of new probers/listeners altogether with *sketches* and *outlines* that might evolve Gym components in order to create methodologies for reproducible research methods in what concerns VNF testing. Moreover, Gym output Profiles can assist the formalization of standard representation methods of VNF testing results, walking in the direction of an envisioned common repository of such data, being open for analysis in a diverse set of applications (e.g., VNF DevOps processes).

7. Acknowledgements

This research was partially supported by the Innovation Center, Ericsson S.A., Brazil, grant UNI.58.

[10]https://wiki.opnfv.org/display/vsperf/VSperf+Home
[11]http://benchmarking.juju.solutions/
[12]https://github.com/toddproject/todd

References

Emmerich, P., Gallenmüller, S., Raumer, D., Wohlfart, F., and Carle, G. (2015). Moongen: A scriptable high-speed packet generator. In *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, pages 275–287, New York, NY, USA. ACM.

ETSI GS NFV-TST (2016). ETSI GS NFV-TST 002 V1.1.1 - Report on NFV Interoperability Testing Methodology.

Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., and Vahdat, A. (2013). B4: Experience with a globally-deployed software defined wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14.

Molnár, L., Pongrácz, G., Enyedi, G., Kis, Z. L., Csikor, L., Juhász, F., Kőrösi, A., and Rétvári, G. (2016). Dataplane specialization for high-performance openflow software switching. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 539–552, New York, NY, USA. ACM.

Morton, A. (2016). Considerations for benchmarking virtual network functions and their infrastructure. IETF BMWG: Internet draft.

Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., and Casado, M. (2015). The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, Oakland, CA. USENIX Association.

Raumer, D., Gallenmüller, S., Wohlfart, F., Emmerich, P., Werneck, P., and Carle, G. (2016). Revisiting benchmarking methodology for interconnect devices. In *Proceedings of the 2016 Applied Networking Research Workshop*, ANRW '16, pages 55–61, New York, NY, USA. ACM.

Rosa, R. V., Rothenberg, C. E., and Szabo, R. (2015a). VBaaS: VNF Benchmark-as-a-Service. In *2015 Fourth European Workshop on Software Defined Networks*, pages 79–84.

Rosa, R. V., Rothenberg, C. E., and Szabo, R. (2015b). VNF Benchmark-as-a-Service. IRTF NFVRG: Internet draft.

Rosa, R. V., Rothenberg, C. E., and Szabo, R. (2016). VNF Benchmarking Methodology. IETF BMWG: Internet draft.

Tahhan, M., O'Mahony, B., and Morton, A. (2016). Benchmarking virtual switches in opnfv. IETF BMWG: Internet draft.