Yardstick Overview

Release draft (5f2938c)

OPNFV

February 10, 2016

CONTENTS

CHAPTER

ONE

INTRODUCTION

Welcome to Yardstick's documentation !

Yardstick is an OPNFV Project.

The project's goal is to verify infrastructure compliance, from the perspective of a *VNF*.

The Project's scope is the development of a test framework, *Yardstick*, test cases and test stimuli to enable *NFVI* verification. The Project also includes a sample *VNF*, the *VTC* and its experimental framework, *ApexLake* !

The chapter Methodology describes the methodology implemented by the Yardstick Project for *NFVI* verification. The chapter Yardstick Test Cases includes a list of available Yardstick test cases.

Yardstick is used for verifying the OPNFV infrastructure and some of the OPNFV features, listed in Yardstick Test Cases.

The *Yardstick* framework is deployed in several OPNFV community labs. It is installer, infrastructure and application independent.

See also:

Pharos for information on OPNFV community labs.

1.1 Contact Yardstick

Feedback? Contact us

CHAPTER TWO

METHODOLOGY

2.1 Abstract

This chapter describes the methodology implemented by the Yardstick project for verifying the NFV Infrastructure from the perspective of a VNF.

2.2 ETSI-NFV

The document ETSI GS NFV-TST001, "Pre-deployment Testing; Report on Validation of NFV Environments and Services", recommends methods for pre-deployment testing of the functional components of an NFV environment.

The Yardstick project implements the methodology described in chapter 6, "Pre- deployment validation of NFV infrastructure".

The methodology consists in decomposing the typical VNF work-load performance metrics into a number of characteristics/performance vectors, which each can be represented by distinct test-cases.

The methodology includes five steps:

- ***Step1:*** **Define Infrastruture - the HW, SW and corresponding configuration** target for validation; the OPNFV infrastructure, in OPNFV community labs.
- ***Step2:*** **Identify VNF type - the application for which the infrastructure is** to be validated, and its requirements on the underlying infrastructure.
- ***Step3:*** **Select test cases - depending on the workload that represents the** application for which the infrastruture is to be validated, the relevant test cases amongst the list of available Yardstick test cases.
- ***Step4:*** **Execute tests - define the duration and number of iterations for the** selected test cases, tests runs are automated via OPNFV Jenkins Jobs.
- *Step5:* Collect results - using the common API for result collection.

2.3 Metrics

The metrics, as defined by ETSI GS NFV-TST001, are shown in *Table1*, *Table2* and *Table3*.

In OPNFV Brahmaputra release, generic test cases covering aspects of the listed metrics are available; further OPNFV releases will provide extended testing of these metrics. The view of available Yardstick test cases cross ETSI definitions in *Table1*, *Table2* and *Table3* is shown in *Table4*. It shall be noticed that the Yardstick test cases are examples, the test duration and number of iterations are configurable, as are the System Under Test (SUT) and the attributes (or, in Yardstick nomemclature, the scenario options). **Table 1 - Performance/Speed Metrics**

Category	Performance/Speed
Compute	• Latency for random memory access • Latency for cache read/write operations • Processing speed (instructions per second) • Throughput for random memory access (bytes per second)
Network	• Throughput per NFVI node (frames/byte per second) • Throughput provided to a VM (frames/byte per second) • Latency per traffic flow • Latency between VMs • Latency between NFVI nodes • Packet delay variation (jitter) between VMs • Packet delay variation (jitter) between NFVI nodes
Storage	• Sequential read/write IOPS • Random read/write IOPS • Latency for storage read/write operations • Throughput for storage read/write operations

Table 2 - Capacity/Scale Metrics

Category	Capacity/Scale
Compute	• Number of cores and threads- Available memory size • Cache size • Processor utilization (max, average, standard deviation) • Memory utilization (max, average, standard deviation) • Cache utilization (max, average, standard deviation)
Network	• Number of connections • Number of frames sent/received • Maximum throughput between VMs (frames/byte per second) • Maximum throughput between NFVI nodes (frames/byte per second) • Network utilization (max, average, standard deviation) • Number of traffic flows
Storage	• Storage/Disk size • Capacity allocation (block-based, object-based) • Block size • Maximum sequential read/write IOPS • Maximum random read/write IOPS • Disk utilization (max, average, standard deviation)

Table 3 - Availability/Reliability Metrics

Category	Availability/Reliability
Compute	• Processor availability (Error free processing time) • Memory availability (Error free memory time) • Processor mean-time-to-failure • Memory mean-time-to-failure • Number of processing faults per second
Network	• NIC availability (Error free connection time) • Link availability (Error free transmission time) • NIC mean-time-to-failure • Network timeout duration due to link failure • Frame loss rate
Storage	• Disk availability (Error free disk access time) • Disk mean-time-to-failure • Number of failed storage read/write operations per second

Table 4 - Yardstick Generic Test Cases

Category	Performance/Speed	Capacity/Scale	Availability/Reliability
Compute	TC003 TC004 TC014 TC024	TC003 TC004 TC010 TC012	TC013 [1] TC015 [1]
Network	TC002 TC011	TC001 TC008 TC009	TC016 [1] TC018 [1]
Storage	TC005	TC005	TC017 [1]

Note: The description in this OPNFV document is intended as a reference for users to understand the scope of the Yardstick Project and the deliverables of the Yardstick framework. For complete description of the methodology, refer to the ETSI document.

[1] To be included in future deliveries.

CHAPTER

THREE

VIRTUAL TRAFFIC CLASSIFIER

3.1 Abstract

This chapter provides an overview of the virtual Traffic Classifier, a contribution to OPNFV Yardstick from the EU Project TNOVA. Additional documentation is available in TNOVAresults.

3.2 Overview

The virtual Traffic Classifier *VNF*, the *VTC*, comprises of a *VNFC*. The *VNFC* contains both the Traffic Inspection module, and the Traffic forwarding module, needed to run the VNF. The exploitation of *DPI* methods for traffic classification is built around two basic assumptions:

- third parties unaffiliated with either source or recipient are able to

inspect each IP packet's payload

- the classifier knows the relevant syntax of each application's packet

payloads (protocol signatures, data patterns, etc.).

The proposed *DPI* based approach will only use an indicative, small number of the initial packets from each flow in order to identify the content and not inspect each packet.

In this respect it follows the *PBFS*. This method uses a table to track each session based on the 5-tuples (src address, dest address, src port,dest port, transport protocol) that is maintained for each flow.

3.3 Concepts

- *Traffic Inspection*: The process of packet analysis and application

identification of network traffic that passes through the *VTC*.

- *Traffic Forwarding*: The process of packet forwarding from an incoming

network interface to a pre-defined outgoing network interface.

- *Traffic Rule Application*: The process of packet tagging, based on a

predefined set of rules. Packet tagging may include e.g. *ToS* field modification.

3.4 Architecture

The Traffic Inspection module is the most computationally intensive component of the *VNF*. It implements filtering and packet matching algorithms in order to support the enhanced traffic forwarding capability of the *VNF*. The component supports a flow table (exploiting hashing algorithms for fast indexing of flows) and an inspection engine for traffic classification.

The implementation used for these experiments exploits the nDPI library. The packet capturing mechanism is implemented using libpcap. When the *DPI* engine identifies a new flow, the flow register is updated with the appropriate information and transmitted across the Traffic Forwarding module, which then applies any required policy updates.

The Traffic Forwarding moudle is responsible for routing and packet forwarding. It accepts incoming network traffic, consults the flow table for classification information for each incoming flow and then applies pre-defined policies marking e.g. *ToS*/*DSCP* multimedia traffic for *QoS* enablement on the forwarded traffic. It is assumed that the traffic is forwarded using the default policy until it is identified and new policies are enforced.

The expected response delay is considered to be negligible, as only a small number of packets are required to identify each flow.

3.5 Graphical Overview

```
+----------------------------+
|                            |
| Virtual Traffic Classifier |
|                            |
|     Analysing/Forwarding   |
|        ------------>       |
|     ethA          ethB     |
|                            |
+----------------------------+
     |              ^
     |              |
     v              |
+----------------------------+
|                            |
|     Virtual Switch         |
|                            |
+----------------------------+
```

3.6 Install

run the build.sh with root privileges

3.7 Run

sudo ./pfbridge -a eth1 -b eth2

3.8 Development Environment

Ubuntu 14.04

CHAPTER

FOUR

YARDSTICK TEST CASES

4.1 Abstract

This chapter lists available Yardstick test cases. Yardstick test cases are divided in two main categories:

- *Generic NFVI Test Cases* - Test Cases developed to realize the methodology

described in Methodology

- *OPNFV Feature Test Cases* - Test Cases developed to verify one or more

aspect of a feature delivered by an OPNFV Project, including the test cases developed for the *VTC*.

4.2 Generic NFVI Test Case Descriptions

4.2.1 Yardstick Test Case Description TC001

Network Performance	
test case id	OPNFV_YARDSTICK_TC001_NW PERF
metric	Number of flows and throughput
test purpose	To evaluate the IaaS network performance with regards to flows and throughput, such as if and how different amounts of flows matter for the throughput between hosts on different compute blades. Typically e.g. the performance of a vSwitch depends on the number of flows running through it. Also performance of other equipment or entities can depend on the number of flows or the packet sizes used. The purpose is also to be able to spot trends. Test results, graphs ans similar shall be stored for comparison reasons and product evolution understanding between different OPNFV versions and/or configurations.
configuration	file: opnfv_yardstick_tc001.yaml Packet size: 60 bytes Number of ports: 10, 50, 100, 500 and 1000, where each runs for 20 seconds. The whole sequence is run twice. The client and server are distributed on different HW. For SLA max_ppm is set to 1000. The amount of configured ports map to between 110 up to 1001000 flows, respectively.
test tool	pktgen (Pktgen is not always part of a Linux distribution, hence it needs to be installed. It is part of the Yardstick Docker image. As an example see the /yardstick/tools/ directory for how to generate a Linux image with pktgen included.)
references	pktgen ETSI-NFV-TST001
applicability	Test can be configured with different packet sizes, amount of flows and test duration. Default values exist. **SLA (optional): max_ppm: The number of packets per million** packets sent that are acceptable to loose, not received.
pre-test conditions	The test case image needs to be installed into Glance with pktgen included in it. No POD specific requirements have been identified.
test sequence	description and expected result
step 1	The hosts are installed, as server and client. pktgen is invoked and logs are produced and stored. Result: Logs are stored.
test verdict	Fails only if SLA is not passed, or if there is a test case execution problem.

4.2.2 Yardstick Test Case Description TC002

Network Latency	
test case id	OPNFV_YARDSTICK_TC002_NW LATENCY
metric	RTT, Round Trip Time
test purpose	To do a basic verification that network latency is within acceptable boundaries when packets travel between hosts located on same or different compute blades. The purpose is also to be able to spot trends. Test results, graphs and similar shall be stored for comparison reasons and product evolution understanding between different OPNFV versions and/or configurations.
configuration	file: opnfv_yardstick_tc002.yaml Packet size 100 bytes. Total test duration 600 seconds. One ping each 10 seconds. SLA RTT is set to maximum 10 ms.
test tool	ping Ping is normally part of any Linux distribution, hence it doesn't need to be installed. It is also part of the Yardstick Docker image. (For example also a Cirros image can be downloaded from cirros-image, it includes ping)
references	Ping man page ETSI-NFV-TST001
applicability	Test case can be configured with different packet sizes, burst sizes, ping intervals and test duration. SLA is optional. The SLA in this test case serves as an example. Considerably lower RTT is expected, and also normal to achieve in balanced L2 environments. However, to cover most configurations, both bare metal and fully virtualized ones, this value should be possible to achieve and acceptable for black box testing. Many real time applications start to suffer badly if the RTT time is higher than this. Some may suffer bad also close to this RTT, while others may not suffer at all. It is a compromise that may have to be tuned for different configuration purposes.
pre-test conditions	The test case image needs to be installed into Glance with ping included in it. No POD specific requirements have been identified.
test sequence	description and expected result
step 1	The hosts are installed, as server and client. Ping is invoked and logs are produced and stored. Result: Logs are stored.
test verdict	Test should not PASS if any RTT is above the optional SLA value, or if there is a test case execution problem.

4.2.3 Yardstick Test Case Description TC005

Storage Performance	
test case id	OPNFV_YARDSTICK_TC005_Storage Performance
metric	IOPS, throughput and latency
test purpose	To evaluate the IaaS storage performance with regards to IOPS, throughput and latency. The purpose is also to be able to spot trends. Test results, graphs and similar shall be stored for comparison reasons and product evolution understanding between different OPNFV versions and/or configurations.
configuration	file: opnfv_yardstick_tc005.yaml IO types: read, write, randwrite, randread, rw IO block size: 4KB, 64KB, 1024KB, where each runs for 30 seconds(10 for ramp time, 20 for runtime). For SLA minimum read/write iops is set to 100, minimum read/write throughput is set to 400 KB/s, and maximum read/write latency is set to 20000 usec.
test tool	fio (fio is not always part of a Linux distribution, hence it needs to be installed. As an example see the /yardstick/tools/ directory for how to generate a Linux image with fio included.)
references	fio ETSI-NFV-TST001
applicability	Test can be configured with different read/write types, IO block size, IO depth, ramp time (runtime required for stable results) and test duration. Default values exist.
pre-test conditions	The test case image needs to be installed into Glance with fio included in it. No POD specific requirements have been identified.
test sequence	description and expected result
step 1	The host is installed and fio is invoked and logs are produced and stored. Result: Logs are stored.
test verdict	Fails only if SLA is not passed, or if there is a test case execution problem.

4.2.4 Yardstick Test Case Description TC008

Packet Loss Extended Test	
test case id	OPNFV_YARDSTICK_TC008_NW PERF, Packet loss Extended Test
metric	Number of flows, packet size and throughput
test purpose	To evaluate the IaaS network performance with regards to flows and throughput, such as if and how different amounts of packet sizes and flows matter for the throughput between VMs on different compute blades. Typically e.g. the performance of a vSwitch depends on the number of flows running through it. Also performance of other equipment or entities can depend on the number of flows or the packet sizes used. The purpose is also to be able to spot trends. Test results, graphs ans similar shall be stored for comparison reasons and product evolution understanding between different OPNFV versions and/or configurations.
configuration	file: opnfv_yardstick_tc008.yaml Packet size: 64, 128, 256, 512, 1024, 1280 and 1518 bytes. Number of ports: 1, 10, 50, 100, 500 and 1000. The amount of configured ports map from 2 up to 1001000 flows, respectively. Each packet_size/port_amount combination is run ten times, for 20 seconds each. Then the next packet_size/port_amount combination is run, and so on. The client and server are distributed on different HW. For SLA max_ppm is set to 1000.
test tool	pktgen (Pktgen is not always part of a Linux distribution, hence it needs to be installed. It is part of the Yardstick Docker image. As an example see the /yardstick/tools/ directory for how to generate a Linux image with pktgen included.)
references	pktgen ETSI-NFV-TST001
applicability	Test can be configured with different packet sizes, amount of flows and test duration. Default values exist. SLA (optional): max_ppm: The number of packets per million packets sent that are acceptable to loose, not received.
pre-test conditions	The test case image needs to be installed into Glance with pktgen included in it. No POD specific requirements have been identified.
test sequence	description and expected result
step 1	The hosts are installed, as server and client. pktgen is invoked and logs are produced and stored. Result: Logs are stored.
test verdict	Fails only if SLA is not passed, or if there is a test case execution problem.

4.2.5 Yardstick Test Case Description TC009

Packet Loss	
test case id	OPNFV_YARDSTICK_TC009_NW PERF, Packet loss
metric	Number of flows, packets lost and throughput
test purpose	To evaluate the IaaS network performance with regards to flows and throughput, such as if and how different amounts of flows matter for the throughput between VMs on different compute blades. Typically e.g. the performance of a vSwitch depends on the number of flows running through it. Also performance of other equipment or entities can depend on the number of flows or the packet sizes used. The purpose is also to be able to spot trends. Test results, graphs ans similar shall be stored for comparison reasons and product evolution understanding between different OPNFV versions and/or configurations.
configuration	file: opnfv_yardstick_tc009.yaml Packet size: 64 bytes Number of ports: 1, 10, 50, 100, 500 and 1000. The amount of configured ports map from 2 up to 1001000 flows, respectively. Each port amount is run ten times, for 20 seconds each. Then the next port_amount is run, and so on. The client and server are distributed on different HW. For SLA max_ppm is set to 1000.
test tool	pktgen (Pktgen is not always part of a Linux distribution, hence it needs to be installed. It is part of the Yardstick Docker image. As an example see the /yardstick/tools/ directory for how to generate a Linux image with pktgen included.)
references	pktgen ETSI-NFV-TST001
applicability	Test can be configured with different packet sizes, amount of flows and test duration. Default values exist. SLA (optional): max_ppm: The number of packets per million packets sent that are acceptable to loose, not received.
pre-test conditions	The test case image needs to be installed into Glance with pktgen included in it. No POD specific requirements have been identified.
test sequence	description and expected result
step 1	The hosts are installed, as server and client. pktgen is invoked and logs are produced and stored. Result: logs are stored.
test verdict	Fails only if SLA is not passed, or if there is a test case execution problem.

4.2.6 Yardstick Test Case Description TC010

Memory Latency	
test case id	OPNFV_YARDSTICK_TC010_Memory Latency
metric	Latency in nanoseconds
test purpose	Measure the memory read latency for varying memory sizes and strides. Whole memory hierarchy is measured including all levels of cache.
configuration	File: opnfv_yardstick_tc010.yaml • SLA (max_latency): 30 nanoseconds • Stride - 128 bytes • Stop size - 64 megabytes • Iterations: 10 - test is run 10 times iteratively. • Interval: 1 - there is 1 second delay between each iteration.
test tool	Lmbench Lmbench is a suite of operating system microbenchmarks. This test uses lat_mem_rd tool from that suite. Lmbench is not always part of a Linux distribution, hence it needs to be installed in the test image
references	man-pages McVoy, Larry W.,and Carl Staelin. “lmbench: Portable Tools for Performance Analysis.” USENIX annual technical conference 1996.
applicability	Test can be configured with different: • strides; • stop_size; • iterations and intervals. There are default values for each above-mentioned option. SLA (optional) : max_latency: The maximum memory latency that is accepted.
pre-test conditions	The test case image needs to be installed into Glance with Lmbench included in the image. No POD specific requirements have been identified.
test sequence	description and expected result
step 1	The host is installed as client. Lmbench’s lat_mem_rd tool is invoked and logs are produced and stored. Result: logs are stored.
test verdict	Test fails if the measured memory latency is above the SLA value or if there is a test case execution problem.

4.2.7 Yardstick Test Case Description TC011

Packet delay variation between VMs	
test case id	OPNFV_YARDSTICK_TC011_Packet delay variation between VMs
metric	jitter: packet delay variation (ms)
test purpose	Measure the packet delay variation sending the packets from one VM to the other.
configuration	File: opnfv_yardstick_tc011.yaml • options: protocol: udp # The protocol used by iperf3 tools bandwidth: 20m # It will send the given number of packets without pausing • runner: duration: 30 # Total test duration 30 seconds. • SLA (optional): jitter: 10 (ms) # The maximum amount of jitter that is accepted.
test tool	iperf3 iPerf3 is a tool for active measurements of the maximum achievable bandwidth on IP networks. It supports tuning of various parameters related to timing, buffers and protocols. The UDP protocols can be used to measure jitter delay. (iperf3 is not always part of a Linux distribution, hence it needs to be installed. It is part of the Yardstick Docker image. As an example see the /yardstick/tools/ directory for how to generate a Linux image with pktgen included.)
references	iperf3 ETSI-NFV-TST001
applicability	Test can be configured with different • **bandwidth: Test case can be configured with different** bandwidth • duration: The test duration can be configured • **jitter: SLA is optional. The SLA in this test case** serves as an example.
pre-test conditions	The test case image needs to be installed into Glance with iperf3 included in the image. No POD specific requirements have been identified.
test sequence	description and expected result
step 1	The hosts are installed, as server and client. iperf3 is invoked and logs are produced and stored. Result: Logs are stored.
test verdict	Test should not PASS if any jitter is above the optional SLA value, or if there is a test case execution problem.

4.2.8 Yardstick Test Case Description TC012

Memory Bandwidth	
test case id	OPNFV_YARDSTICK_TC012_Memory Bandwidth
metric	Megabyte per second (MBps)
test purpose	Measure the rate at which data can be read from and written to the memory (this includes all levels of memory).
configuration	File: opnfv_yardstick_tc012.yaml • SLA (optional): 15000 (MBps) min_bw: The minimum amount of memory bandwidth that is accepted. • Size: 10 240 kB - test allocates twice that size (20 480kB) zeros it and then measures the time it takes to copy from one side to another. • Benchmark: rdwr - measures the time to read data into memory and then write data to the same location. • Warmup: 0 - the number of iterations to perform before taking actual measurements. • Iterations: 10 - test is run 10 times iteratively. • Interval: 1 - there is 1 second delay between each iteration.
test tool	Lmbench Lmbench is a suite of operating system microbenchmarks. This test uses bw_mem tool from that suite. Lmbench is not always part of a Linux distribution, hence it needs to be installed in the test image.
references	man-pages McVoy, Larry W., and Carl Staelin. “lmbench: Portable Tools for Performance Analysis.” USENIX annual technical conference. 1996.
applicability	Test can be configured with different: • memory sizes; • memory operations (such as rd, wr, rdwr, cp, frd, fwr, fcp, bzero, bcopy); • number of warmup iterations; • iterations and intervals. There are default values for each above-mentioned option.
pre-test conditions	The test case image needs to be installed into Glance with Lmbench included in the image. No POD specific requirements have been identified.
test sequence	description and expected result
step 1	The host is installed as client. Lmbench’s bw_mem tool is invoked and logs are produced and stored. Result: logs are stored.
test verdict	Test fails if the measured memory bandwidth is below the SLA value or if there is a test case execution problem.

4.2.9 Yardstick Test Case Description TC014

Processing speed	
test case id	OPNFV_YARDSTICK_TC014_Processing speed
metric	score of single cpu running, score of parallel running
test purpose	To evaluate the IaaS processing speed with regards to score of single cpu running and parallel running The purpose is also to be able to spot trends. Test results, graphs and similar shall be stored for comparison reasons and product evolution understanding between different OPNFV versions and/or configurations.
configuration	file: opnfv_yardstick_tc014.yaml run_mode: Run unixbench in quiet mode or verbose mode test_type: dhry2reg, whetstone and so on For SLA with single_score and parallel_score, both can be set by user, default is NA
test tool	unixbench (unixbench is not always part of a Linux distribution, hence it needs to be installed. As an example see the /yardstick/tools/ directory for how to generate a Linux image with unixbench included.)
references	unixbench ETSI-NFV-TST001
applicability	Test can be configured with different test types, dhry2reg, whetstone and so on.
pre-test conditions	The test case image needs to be installed into Glance with unixbench included in it. No POD specific requirements have been identified.
test sequence	description and expected result
step 1	The hosts are installed, as a client. unixbench is invoked and logs are produced and stored. Result: Logs are stored.
test verdict	Fails only if SLA is not passed, or if there is a test case execution problem.

4.2.10 Yardstick Test Case Description TC024

CPU Load	
test case id	OPNFV_YARDSTICK_TC024_CPU Load
metric	CPU load
test purpose	To evaluate the CPU load performance of the IaaS. This test case should be run in parallel to other Yardstick test cases and not run as a stand-alone test case. The purpose is also to be able to spot trends. Test results, graphs ans similar shall be stored for comparison reasons and product evolution understanding between different OPNFV versions and/or configurations.
configuration	file: cpuload.yaml (in the 'samples' directory) There is are no additional configurations to be set for this TC.
test tool	mpstat (mpstat is not always part of a Linux distribution, hence it needs to be installed. It is part of the Yardstick Glance image. However, if mpstat is not present the TC instead uses /proc/stats as source to produce "mpstat" output.
references	man-pages
applicability	Run in background with other test cases.
pre-test conditions	The test case image needs to be installed into Glance with mpstat included in it. No POD specific requirements have been identified.
test sequence	description and expected result
step 1	The host is installed. The related TC, or TCs, is invoked and mpstat logs are produced and stored. Result: Stored logs
test verdict	None. CPU load results are fetched and stored.

4.2.11 Yardstick Test Case Description TC037

Latency, CPU Load, Throughput, Packet Loss	
test case id	OPNFV_YARDSTICK_TC037_Latency,CPU Load,Throughput,Packet Loss
metric	Number of flows, latency, throughput, CPU load, packet loss
test purpose	To evaluate the IaaS network performance with regards to flows and throughput, such as if and how different amounts of flows matter for the throughput between hosts on different compute blades. Typically e.g. the performance of a vSwitch depends on the number of flows running through it. Also performance of other equipment or entities can depend on the number of flows or the packet sizes used. The purpose is also to be able to spot trends. Test results, graphs ans similar shall be stored for comparison reasons and product evolution understanding between different OPNFV versions and/or configurations.
configuration	file: opnfv_yardstick_tc037.yaml Packet size: 64 bytes Number of ports: 1, 10, 50, 100, 300, 500, 750 and 1000. The amount configured ports map from 2 up to 1001000 flows, respectively. Each port amount is run two times, for 20 seconds each. Then the next port_amount is run, and so on. During the test CPU load on both client and server, and the network latency between the client and server are measured. The client and server are distributed on different HW. For SLA max_ppm is set to 1000.
test tool	pktgen (Pktgen is not always part of a Linux distribution, hence it needs to be installed. It is part of the Yardstick Glance image. As an example see the /yardstick/tools/ directory for how to generate a Linux image with pktgen included.) ping Ping is normally part of any Linux distribution, hence it doesn't need to be installed. It is also part of the Yardstick Glance image. (For example also a cirros image can be downloaded, it includes ping) mpstat (Mpstat is not always part of a Linux distribution, hence it needs to be installed. It is part of the Yardstick Glance image.
references	Ping and Mpstat man pages pktgen ETSI-NFV-TST001
applicability	Test can be configured with different packet sizes, amount of flows and test duration. Default values exist. SLA (optional): max_ppm: The number of packets per million packets sent that are acceptable to loose, not received.
pre-test conditions	The test case image needs to be installed into Glance with pktgen included in it. No POD specific requirements have been identified.
test sequence	description and expected result
step 1	The hosts are installed, as server and client. pktgen is invoked and logs are produced and stored. Result: Logs are stored.
test verdict	Fails only if SLA is not passed, or if there is a test case execution problem.

4.2.12 Yardstick Test Case Description TC038

Latency, CPU Load, Throughput, Packet Loss (Extended measurements)	
test case id	OPNFV_YARDSTICK_TC038_Latency,CPU Load,Throughput,Packet Loss
metric	Number of flows, latency, throughput, CPU load, packet loss
test purpose	To evaluate the IaaS network performance with regards to flows and throughput, such as if and how different amounts of flows matter for the throughput between hosts on different compute blades. Typically e.g. the performance of a vSwitch depends on the number of flows running through it. Also performance of other equipment or entities can depend on the number of flows or the packet sizes used. The purpose is also to be able to spot trends. Test results, graphs ans similar shall be stored for comparison reasons and product evolution understanding between different OPNFV versions and/or configurations.
configuration	file: opnfv_yardstick_tc038.yaml Packet size: 64 bytes Number of ports: 1, 10, 50, 100, 300, 500, 750 and 1000. The amount configured ports map from 2 up to 1001000 flows, respectively. Each port amount is run ten times, for 20 seconds each. Then the next port_amount is run, and so on. During the test CPU load on both client and server, and the network latency between the client and server are measured. The client and server are distributed on different HW. For SLA max_ppm is set to 1000.
test tool	pktgen (Pktgen is not always part of a Linux distribution, hence it needs to be installed. It is part of the Yardstick Glance image. As an example see the /yardstick/tools/ directory for how to generate a Linux image with pktgen included.) ping Ping is normally part of any Linux distribution, hence it doesn't need to be installed. It is also part of the Yardstick Glance image. (For example also a cirros image can be downloaded, it includes ping) mpstat (Mpstat is not always part of a Linux distribution, hence it needs to be installed. It is part of the Yardstick Glance image.
references	Ping and Mpstat man pages pktgen ETSI-NFV-TST001
applicability	Test can be configured with different packet sizes, amount of flows and test duration. Default values exist. SLA (optional): max_ppm: The number of packets per million packets sent that are acceptable to loose, not received.
pre-test conditions	The test case image needs to be installed into Glance with pktgen included in it. No POD specific requirements have been identified.
test sequence	description and expected result
step 1	The hosts are installed, as server and client. pktgen is invoked and logs are produced and stored. Result: Logs are stored.
test verdict	Fails only if SLA is not passed, or if there is a test case execution problem.

4.3 OPNFV Feature Test Cases

4.3.1 H A

Yardstick Test Case Description TC019

Control Node Openstack Service High Availability	
test case id	OPNFV_YARDSTICK_TC019_HA: Control node Openstack service down
test purpose	This test case will verify the high availability of the service provided by OpenStack (like nova-api, neutro-server) on control node.
test method	This test case kills the processes of a specific Openstack service on a selected control node, then checks whether the request of the related Openstack command is OK and the killed processes are recovered.
attackers	In this test case, an attacker called "kill-process" is needed. This attacker includes three parameters: 1) fault_type: which is used for finding the attacker's scripts. It should be always set to "kill-process" in this test case. 2) process_name: which is the process name of the specified OpenStack service. If there are multiple processes use the same name on the host, all of them are killed by this attacker. 3) host: which is the name of a control node being attacked. e.g. -fault_type: "kill-process" -process_name: "nova-api" -host: node1
monitors	In this test case, two kinds of monitor are needed: 1. the "openstack-cmd" monitor constantly request a specific Openstack command, which needs two parameters: 1) monitor_type: which is used for finding the monitor class and related scritps. It should be always set to "openstack-cmd" for this monitor. 2) command_name: which is the command name used for request 2. the "process" monitor check whether a process is running on a specific node, which needs three parameters: 1) monitor_type: which used for finding the monitor class and related scritps. It should be always set to "process" for this monitor. 2) process_name: which is the process name for monitor 3) host: which is the name of the node runing the process e.g. monitor1: -monitor_type: "openstack-cmd" -command_name: "nova image-list" monitor2: -monitor_type: "process" -process_name: "nova-api" -host: node1
metrics	In this test case, there are two metrics: 1)service_outage_time: which indicates the maximum outage time (seconds) of the specified Openstack command request. 2)process_recover_time: which indicates the maximun time (seconds) from the process being killed to recovered
test tool	Developed by the project. Please see folder: "yardstick/benchmark/scenarios/availability/ha_tools"
references	ETSI NFV REL001
configuration	This test case needs two configuration files: 1) test case file: opnfv_yardstick_tc019.yaml -Attackers: see above "attackers" discription -waiting_time: which is the time

Yardstick Test Case Description TC025

OpenStack Controller Node abnormally shutdown High Availability	
test case id	OPNFV_YARDSTICK_TC025_HA: OpenStack Controller Node abnormally shutdown
test purpose	This test case will verify the high availability of controller node. When one of the controller node abnormally shutdown, the service provided by it should be OK.
test method	This test case shutdowns a specified controller node with some fault injection tools, then checks whether all services provided by the controller node are OK with some monitor tools.
attackers	In this test case, an attacker called “host-shutdown” is needed. This attacker includes two parameters: 1) fault_type: which is used for finding the attacker’s scripts. It should be always set to “host-shutdown” in this test case. 2) host: the name of a controller node being attacked. e.g. -fault_type: “host-shutdown” -host: node1
monitors	In this test case, one kind of monitor are needed: 1. the “openstack-cmd” monitor constantly request a specific Openstack command, which needs two parameters 1) monitor_type: which is used for finding the monitor class and related scritps. It should be always set to “openstack-cmd” for this monitor. 2) command_name: which is the command name used for request There are four instance of the “openstack-cmd” monitor: monitor1: -monitor_type: “openstack-cmd” -api_name: “nova image-list” monitor2: -monitor_type: “openstack-cmd” -api_name: “neutron router-list” monitor3: -monitor_type: “openstack-cmd” -api_name: “heat stack-list” monitor4: -monitor_type: “openstack-cmd” -api_name: “cinder list”
metrics	In this test case, there is one metric: 1)service_outage_time: which indicates the maximum outage time (seconds) of the specified Openstack command request.
test tool	Developed by the project. Please see folder: “yardstick/benchmark/scenarios/availability/ha_tools”
references	ETSI NFV REL001
configuration	This test case needs two configuration files: 1) test case file: opnfv_yardstick_tc019.yaml -Attackers: see above “attackers” discription -waiting_time: which is the time (seconds) from the process being killed to stoping monitors the monitors -Monitors: see above “monitors” discription -SLA: see above “metrics” discription 2)POD file: pod.yaml The POD configuration should record on pod.yaml first. the “host” item in this test case will use the node name in the pod.yaml.
test sequence	description and expected result
step 1	start monitors: each monitor will run with independently process Result: The monitor info will be collected.
step 2	do attacker: connect the host through SSH, and then execute shutdown script on the host Result: The host will be shutdown.
step 3	stop monitors after a period of time specified by “waiting_time”

4.3.2 IPv6

Yardstick Test Case Description TC027

IPv6 connectivity between nodes on the tenant network	
test case id	OPNFV_YARDSTICK_TC002_IPv6 connectivity
metric	RTT, Round Trip Time
test purpose	To do a basic verification that IPv6 connectivity is within acceptable boundaries when ipv6 packets travel between hosts located on same or different compute blades. The purpose is also to be able to spot trends. Test results, graphs and similar shall be stored for comparison reasons and product evolution understanding between different OPNFV versions and/or configurations.
configuration	file: opnfv_yardstick_tc027.yaml Packet size 56 bytes. SLA RTT is set to maximum 10 ms.
test tool	ping6 Ping6 is normally part of Linux distribution, hence it doesn't need to be installed.
references	ipv6 ETSI-NFV-TST001
applicability	Test case can be configured with different run step you can run setup, run benchmakr, teardown independently SLA is optional. The SLA in this test case serves as an example. Considerably lower RTT is expected.
pre-test conditions	The test case image needs to be installed into Glance with ping6 included in it. No POD specific requirements have been identified.
test sequence	description and expected result
step 1	The hosts are installed, as server and client. Ping is invoked and logs are produced and stored. Result: Logs are stored.
test verdict	Test should not PASS if any RTT is above the optional SLA value, or if there is a test case execution problem.

4.3.3 KVM

Yardstick Test Case Description TC028

KVM Latency measurements	
test case id	OPNFV_YARDSTICK_TC028_KVM Latency measurements
metric	min, avg and max latency
test purpose	To evaluate the IaaS KVM virtualization capability with regards to min, avg and max latency. The purpose is also to be able to spot trends. Test results, graphs and similar shall be stored for comparison reasons and product evolution understanding between different OPNFV versions and/or configurations.
configuration	file: samples/cyclictest-node-context.yaml
test tool	Cyclictest (Cyclictest is not always part of a Linux distribution, hence it needs to be installed. As an example see the /yardstick/tools/ directory for how to generate a Linux image with cyclictest included.)
references	Cyclictest
applicability	This test case is mainly for kvm4nfv project CI verify. Upgrade host linux kernel, boot a gust vm update it's linux kernel, and then run the cyclictest to test the new kernel is work well.
pre-test conditions	The test kernel rpm, test sequence scripts and test guest image need put the right folders as specified in the test case yaml file. The test guest image needs with cyclictest included in it. No POD specific requirements have been identified.
test sequence	description and expected result
step 1	The host and guest os kernel is upgraded. Cyclictest is invoked and logs are produced and stored. Result: Logs are stored.
test verdict	Fails only if SLA is not passed, or if there is a test case execution problem.

4.3.4 Parser

Yardstick Test Case Description TC040

Verify Parser Yang-to-Tosca	
test case id	OPNFV_YARDSTICK_TC040 Verify Parser Yang-to-Tosca
metric	1. tosca file which is converted from yang file by Parser 2. result whether the output is same with expected outcome
test purpose	To verify the function of Yang-to-Tosca in Parser.
configuration	file: opnfv_yardstick_tc040.yaml yangfile: the path of the yangfile which you want to convert toscafile: the path of the toscafile which is your expected outcome.
test tool	Parser (Parser is not part of a Linux distribution, hence it needs to be installed. As an example see the /yardstick/benchmark/scenarios/parser/parser_setup.sh for how to install it manual. Of course, it will be installed and uninstalled automatically when you run this test case by yardstick)
references	Parser
applicability	Test can be configured with different path of yangfile and toscafile to fit your real environment to verify Parser
pre-test conditions	No POD specific requirements have been identified. it can be run without VM
test sequence	description and expected result
step 1	parser is installed without VM, running Yang-to-Tosca module to convert yang file to tosca file, validating output against expected outcome. Result: Logs are stored.
test verdict	Fails only if output is different with expected outcome or if there is a test case execution problem.

4.3.5 virtual Traffic Classifier

Yardstick Test Case Description TC006

Network Performance	
test case id	OPNFV_YARDSTICK_TC006_Virtual Traffic Classifier Data Plane Throughput Benchmarking Test.
metric	Throughput
test purpose	To measure the throughput supported by the virtual Traffic Classifier according to the RFC2544 methodology for a user-defined set of vTC deployment configurations.
configuration	file: file: opnfv_yardstick_tc006.yaml **packet_size: size of the packets to be used during the** throughput calculation. Allowe values: [64, 128, 256, 512, 1024, 1280, 1518] **vnic_type: type of VNIC to be used.** **Allowed values are:** • normal: for default OvS port configuration • direct: for SR-IOV port configuration Default value: None **vtc_flavor: OpenStack flavor to be used for the vTC** Default available values are: m1.small, m1.medium, and m1.large, but the user can create his/her own flavor and give it as input Default value: None **vlan_sender: vlan tag of the network on which the vTC will** receive traffic (VLAN Network 1). Allowed values: range (1, 4096) **vlan_receiver: vlan tag of the network on which the vTC** will send traffic back to the packet generator (VLAN Network 2). Allowed values: range (1, 4096) **default_net_name: neutron name of the defaul network that** is used for access to the internet from the vTC (vNIC 1). **default_subnet_name: subnet name for vNIC1** (information available through Neutron). **vlan_net_1_name: Neutron Name for VLAN Network 1** (information available through Neutron). **vlan_subnet_1_name: Subnet Neutron name for VLAN Network 1** (information available through Neutron). **vlan_net_2_name: Neutron Name for VLAN Network 2** (information available through Neutron). **vlan_subnet_2_name: Subnet Neutron name for VLAN Network 2** (information available through Neutron).
test tool	DPDK pktgen DPDK Pktgen is not part of a Linux distribution, hence it needs to be installed by the user.
references	DPDK Pktgen: DPDKpktgen ETSI-NFV-TST001 RFC 2544: rfc2544
applicability	Test can be configured with different flavors, vNIC type and packet sizes. Default values exist as specified above. The vNIC type and flavor MUST be specified by the user.
pre-test	The vTC has been successfully instantiated and configured. The user has correctly assigned the values to the deployment

Yardstick Test Case Description TC007

Network Performance	
test case id	**OPNFV_YARDSTICK_TC007_Virtual Traffic Classifier Data Plane** Throughput Benchmarking Test in Presence of Noisy neighbours
metric	Throughput
test purpose	To measure the throughput supported by the virtual Traffic Classifier according to the RFC2544 methodology for a user-defined set of vTC deployment configurations in the presence of noisy neighbours.
configuration	file: opnfv_yardstick_tc007.yaml **packet_size: size of the packets to be used during the** throughput calculation. Allowe values: [64, 128, 256, 512, 1024, 1280, 1518] **vnic_type: type of VNIC to be used.** **Allowed values are:** • normal: for default OvS port configuration • direct: for SR-IOV port configuration **vtc_flavor: OpenStack flavor to be used for the vTC** Default available values are: m1.small, m1.medium, and m1.large, but the user can create his/her own flavor and give it as input **num_of_neighbours: Number of noisy neighbours (VMs) to be** instantiated during the experiment. Allowed values: range (1, 10) **amount_of_ram: RAM to be used by each neighbor.** **Allowed values: ['250M', '1G', '2G', '3G', '4G', '5G',** '6G', '7G', '8G', '9G', '10G'] Deault value: 256M **number_of_cores: Number of noisy neighbours (VMs) to be** instantiated during the experiment. Allowed values: range (1, 10) Default value: 1 **vlan_sender: vlan tag of the network on which the vTC will** receive traffic (VLAN Network 1). Allowed values: range (1, 4096) **vlan_receiver: vlan tag of the network on which the vTC** will send traffic back to the packet generator (VLAN Network 2). Allowed values: range (1, 4096) **default_net_name: neutron name of the defaul network that** is used for access to the internet from the vTC (vNIC 1). **default_subnet_name: subnet name for vNIC1** (information available through Neutron). **vlan_net_1_name: Neutron Name for VLAN Network 1** (information available through Neutron). **vlan_subnet_1_name: Subnet Neutron name for VLAN Network 1** (information available through Neutron). **vlan_net_2_name: Neutron Name for VLAN Network 2** (information available through Neutron). **vlan_subnet_2_name: Subnet Neutron name for VLAN Network 2** (information available through Neutron).
test tool	DPDK pktgen DPDK Pktgen is not part of a Linux distribution, hence

Yardstick Test Case Description TC020

Network Performance	
test case id	OPNFV_YARDSTICK_TC0020_Virtual Traffic Classifier Instantiation Test
metric	Failure
test purpose	To verify that a newly instantiated vTC is ‘alive’ and functional and its instantiation is correctly supported by the infrastructure.
configuration	file: opnfv_yardstick_tc020.yaml **vnic_type: type of VNIC to be used.** **Allowed values are:** • normal: for default OvS port configuration • direct: for SR-IOV port configuration Default value: None **vtc_flavor: OpenStack flavor to be used for the vTC** Default available values are: m1.small, m1.medium, and m1.large, but the user can create his/her own flavor and give it as input Default value: None **vlan_sender: vlan tag of the network on which the vTC will** receive traffic (VLAN Network 1). Allowed values: range (1, 4096) **vlan_receiver: vlan tag of the network on which the vTC** will send traffic back to the packet generator (VLAN Network 2). Allowed values: range (1, 4096) **default_net_name: neutron name of the defaul network that** is used for access to the internet from the vTC (vNIC 1). **default_subnet_name: subnet name for vNIC1** (information available through Neutron). **vlan_net_1_name: Neutron Name for VLAN Network 1** (information available through Neutron). **vlan_subnet_1_name: Subnet Neutron name for VLAN Network 1** (information available through Neutron). **vlan_net_2_name: Neutron Name for VLAN Network 2** (information available through Neutron). **vlan_subnet_2_name: Subnet Neutron name for VLAN Network 2** (information available through Neutron).
test tool	DPDK pktgen DPDK Pktgen is not part of a Linux distribution, hence it needs to be installed by the user.
references	DPDKpktgen ETSI-NFV-TST001 rfc2544
applicability	Test can be configured with different flavors, vNIC type and packet sizes. Default values exist as specified above. The vNIC type and flavor MUST be specified by the user.
pre-test	The vTC has been successfully instantiated and configured. The user has correctly assigned the values to the deployment configuration parameters. • **Multicast traffic MUST be enabled on the network.** The Data network switches need to be configured in order to manage multicast traffic. Installation and configuration of smcroute

Yardstick Test Case Description TC021

Network Performance	
test case id	OPNFV_YARDSTICK_TC0021_Virtual Traffic Classifier Instantiation Test in Presence of Noisy Neighbours
metric	Failure
test purpose	To verify that a newly instantiated vTC is 'alive' and functional and its instantiation is correctly supported by the infrastructure in the presence of noisy neighbours.
configuration	file: opnfv_yardstick_tc021.yaml **vnic_type: type of VNIC to be used.** **Allowed values are:** • normal: for default OvS port configuration • direct: for SR-IOV port configuration Default value: None **vtc_flavor: OpenStack flavor to be used for the vTC** Default available values are: m1.small, m1.medium, and m1.large, but the user can create his/her own flavor and give it as input Default value: None **num_of_neighbours: Number of noisy neighbours (VMs) to be** instantiated during the experiment. Allowed values: range (1, 10) **amount_of_ram: RAM to be used by each neighbor.** **Allowed values: ['250M', '1G', '2G', '3G', '4G', '5G',** '6G', '7G', '8G', '9G', '10G'] Deault value: 256M **number_of_cores: Number of noisy neighbours (VMs) to be** instantiated during the experiment. Allowed values: range (1, 10) Default value: 1 **vlan_sender: vlan tag of the network on which the vTC will** receive traffic (VLAN Network 1). Allowed values: range (1, 4096) **vlan_receiver: vlan tag of the network on which the vTC** will send traffic back to the packet generator (VLAN Network 2). Allowed values: range (1, 4096) **default_net_name: neutron name of the defaul network that** is used for access to the internet from the vTC (vNIC 1). **default_subnet_name: subnet name for vNIC1** (information available through Neutron). **vlan_net_1_name: Neutron Name for VLAN Network 1** (information available through Neutron). **vlan_subnet_1_name: Subnet Neutron name for VLAN Network 1** (information available through Neutron). **vlan_net_2_name: Neutron Name for VLAN Network 2** (information available through Neutron). **vlan_subnet_2_name: Subnet Neutron name for VLAN Network 2** (information available through Neutron).
test tool	DPDK pktgen DPDK Pktgen is not part of a Linux distribution, hence it needs to be installed by the user.
references	DPDK Pktgen: DPDK Pktgen: DPDKpktgen ETSI-NFV-TST001 RFC 2544: rfc2544
applicability	Test can be configured with different flavors, vNIC type

4.4 Templates

4.4.1 Yardstick Test Case Description TCXXX

test case slogan e.g. Network Latency	
test case id	e.g. OPNFV_YARDSTICK_TC001_NW Latency
metric	what will be measured, e.g. latency
test purpose	describe what is the purpose of the test case
configuration	what .yaml file to use, state SLA if applicable, state test duration, list and describe the scenario options used in this TC and also list the options using default values.
test tool	e.g. ping
references	e.g. RFCxxx, ETSI-NFVyyy
applicability	describe variations of the test case which can be performend, e.g. run the test for different packet sizes
pre-test conditions	describe configuration in the tool(s) used to perform the measurements (e.g. fio, pktgen), POD-specific configuration required to enable running the test
test sequence	description and expected result
step 1	use this to describe tests that require sveveral steps e.g collect logs. Result: what happens in this step e.g. logs collected
step 2	remove interface Result: interface down.
step N	what is done in step N Result: what happens
test verdict	expected behavior, or SLA, pass/fail criteria

4.4.2 Task Template Syntax

Basic template syntax

A nice feature of the input task format used in Yardstick is that it supports the template syntax based on Jinja2. This turns out to be extremely useful when, say, you have a fixed structure of your task but you want to parameterize this task in some way. For example, imagine your input task file (task.yaml) runs a set of Ping scenarios:

```
# Sample benchmark task config file
# measure network latency using ping
schema: "yardstick:task:0.1"

scenarios:
-
  type: Ping
  options:
    packetsize: 200
  host: athena.demo
  target: ares.demo

  runner:
    type: Duration
    duration: 60
    interval: 1

  sla:
    max_rtt: 10
    action: monitor

context:
    ...
```

Let's say you want to run the same set of scenarios with the same runner/ context/sla, but you want to try another packetsize to compare the performance. The most elegant solution is then to turn the packetsize name into a template variable:

```
# Sample benchmark task config file
# measure network latency using ping

schema: "yardstick:task:0.1"
scenarios:
-
  type: Ping
  options:
    packetsize: {{packetsize}}
  host: athena.demo
  target: ares.demo

  runner:
    type: Duration
    duration: 60
    interval: 1

  sla:
    max_rtt: 10
    action: monitor

context:
    ...
```

and then pass the argument value for {{packetsize}} when starting a task with this configuration file. Yardstick provides you with different ways to do that:

1.Pass the argument values directly in the command-line interface (with either a JSON or YAML dictionary):

```
yardstick task start samples/ping-template.yaml
--task-args'{"packetsize":"200"}'
```

2.Refer to a file that specifies the argument values (JSON/YAML):

```
yardstick task start samples/ping-template.yaml --task-args-file args.yaml
```

Using the default values

Note that the Jinja2 template syntax allows you to set the default values for your parameters. With default values set, your task file will work even if you don't parameterize it explicitly while starting a task. The default values should be set using the {% set ... %} clause (task.yaml). For example:

```
# Sample benchmark task config file
# measure network latency using ping
schema: "yardstick:task:0.1"
{% set packetsize = packetsize or "100" %}
scenarios:
-
```

```
  type: Ping
  options:
  packetsize: {{packetsize}}
  host: athena.demo
  target: ares.demo

  runner:
    type: Duration
    duration: 60
    interval: 1
  ...
```

If you don't pass the value for {{packetsize}} while starting a task, the default one will be used.

Advanced templates

Yardstick makes it possible to use all the power of Jinja2 template syntax, including the mechanism of built-in functions. As an example, let us make up a task file that will do a block storage performance test. The input task file (fio-template.yaml) below uses the Jinja2 for-endfor construct to accomplish that:

```
#Test block sizes of 4KB, 8KB, 64KB, 1MB
#Test 5 workloads: read, write, randwrite, randread, rw
schema: "yardstick:task:0.1"

 scenarios:
{% for bs in ['4k', '8k', '64k', '1024k' ] %}
  {% for rw in ['read', 'write', 'randwrite', 'randread', 'rw' ] %}
-
  type: Fio
  options:
    filename: /home/ubuntu/data.raw
    bs: {{bs}}
    rw: {{rw}}
    ramp_time: 10
  host: fio.demo
  runner:
    type: Duration
    duration: 60
    interval: 60

  {% endfor %}
{% endfor %}
context
    ...
```

CHAPTER

FIVE

YARDSTICK GLOSSARY

DPI	Deep Packet Inspection
DSCP	Differentiated Services Code Point
NFVI	Network Function Virtualization Infrastructure
PBFS	Packet Based per Flow State
QoS	Quality of Service
ToS	Type of Service
VNF	Virtual Network Function
VNFC	Virtual Network Function Component
VTC	Virtual Traffic Classifier