



Yardstick Overview

Release draft (9d041d6)

OPNFV

May 17, 2016

| | | |
|----------|--|-----------|
| 1 | Introduction | 1 |
| 1.1 | About This Document | 1 |
| 1.2 | Contact Yardstick | 1 |
| 2 | Methodology | 3 |
| 2.1 | Abstract | 3 |
| 2.2 | ETSI-NFV | 3 |
| 2.3 | Metrics | 3 |
| 3 | Architecture | 7 |
| 3.1 | Abstract | 7 |
| 3.2 | Overview | 7 |
| 3.3 | Use-Case View | 8 |
| 3.4 | Logical View | 9 |
| 3.5 | Process View (Test execution flow) | 10 |
| 3.6 | Deployment View | 11 |
| 3.7 | Yardstick Directory structure | 12 |
| 4 | Virtual Traffic Classifier | 13 |
| 4.1 | Abstract | 13 |
| 4.2 | Overview | 13 |
| 4.3 | Concepts | 13 |
| 4.4 | Architecture | 14 |
| 4.5 | Graphical Overview | 14 |
| 4.6 | Install | 14 |
| 4.7 | Run | 14 |
| 4.8 | Development Environment | 15 |
| 5 | Apexlake Installation Guide | 17 |
| 5.1 | Abstract | 17 |
| 6 | Apexlake API Interface Definition | 21 |
| 6.1 | Abstract | 21 |
| 6.2 | init | 21 |
| 6.3 | execute_framework | 21 |
| 7 | Yardstick Installation | 23 |
| 7.1 | Abstract | 23 |
| 7.2 | Installing Yardstick on Ubuntu 14.04 | 23 |
| 7.3 | Installing Yardstick using Docker | 24 |
| 7.4 | OpenStack parameters and credentials | 25 |

| | | |
|-----------|---|-----------|
| 7.5 | Examples and verifying the install | 26 |
| 8 | Yardstick Test Cases | 27 |
| 8.1 | Abstract | 27 |
| 8.2 | Generic NFVI Test Case Descriptions | 28 |
| 8.3 | OPNFV Feature Test Cases | 41 |
| 8.4 | Templates | 55 |
| 9 | Glossary | 59 |
| 10 | References | 61 |
| 10.1 | OPNFV | 61 |
| 10.2 | References used in Test Cases | 61 |
| 10.3 | Research | 62 |
| 10.4 | Standards | 62 |
| | Index | 63 |

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 Virtual Network Function (*VNF*).

The Project's scope is the development of a test framework, *Yardstick*, test cases and test stimuli to enable Network Function Virtualization Infrastructure (*NFVI*) verification. The Project also includes a sample *VNF*, the Virtual Traffic Classifier (*VTC*) and its experimental framework, *ApexLake* !

Yardstick is used in OPNFV for verifying the OPNFV infrastructure and some of the OPNFV features. 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 and this [Presentation](#) for an overview of *Yardstick*

1.1 About This Document

This document consists of the following chapters:

- Chapter [Methodology](#) describes the methodology implemented by the Yardstick Project for *NFVI* verification.
- Chapter [Architecture](#) provides information on the software architecture of yardstick.
- Chapter [Virtual Traffic Classifier](#) provides information on the *VTC*.
- Chapter [Apexlake Installation Guide](#) provides instructions to install the experimental framework *ApexLake* and chapter [Apexlake API Interface Definition](#) explains how this framework is integrated in *Yardstick*.
- Chapter [Yardstick Installation](#) provides instructions to install *Yardstick*.
- Chapter [Yardstick Test Cases](#) includes a list of available Yardstick test cases.

1.2 Contact Yardstick

Feedback? [Contact us](#)

METHODOLOGY

2.1 Abstract

This chapter describes the methodology implemented by the Yardstick project for verifying the *NFVI* 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 Infrastructure - the Hardware, Software 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 infrastructure 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.

See also:

[Yardsticktst](#) for material on alignment ETSI TST001 and Yardstick.

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 nomenclature, the scenario options). **Table 1 - Performance/Speed Metrics**

| Category | Performance/Speed |
|----------|---|
| Compute | <ul style="list-style-type: none"> • 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 | <ul style="list-style-type: none"> • 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 | <ul style="list-style-type: none"> • 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 | <ul style="list-style-type: none"> • 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 | <ul style="list-style-type: none"> • 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 | <ul style="list-style-type: none"> • 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 | <ul style="list-style-type: none"> • 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 | <ul style="list-style-type: none"> • 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 | <ul style="list-style-type: none"> • 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 ¹ TC004 ¹ TC014 TC024 | TC003 ¹ TC004 ¹ 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.

¹To be included in future deliveries.

ARCHITECTURE

3.1 Abstract

This chapter describes the yardstick framework software architecture. we will introduce it from Use-Case View, Logical View, Process View and Deployment View. More technical details will be introduced in this chapter.

3.2 Overview

3.2.1 Architecture overview

Yardstick is mainly written in Python, and test configurations are made in YAML. Documentation is written in re-StructuredText format, i.e. .rst files. Yardstick is inspired by Rally. Yardstick is intended to run on a computer with access and credentials to a cloud. The test case is described in a configuration file given as an argument.

How it works: the benchmark task configuration file is parsed and converted into an internal model. The context part of the model is converted into a Heat template and deployed into a stack. Each scenario is run using a runner, either serially or in parallel. Each runner runs in its own subprocess executing commands in a VM using SSH. The output of each scenario is written as json records to a file or influxdb or http server, we use influxdb as the backend, the test result will be shown with grafana.

3.2.2 Concept

Benchmark - assess the relative performance of something

Benchmark configuration file - describes a single test case in yaml format

Context - The set of Cloud resources used by a scenario, such as user names, image names, affinity rules and network configurations. A context is converted into a simplified Heat template, which is used to deploy onto the Openstack environment.

Data - Output produced by running a benchmark, written to a file in json format

Runner - Logic that determines how a test scenario is run and reported, for example the number of test iterations, input value stepping and test duration. Predefined runner types exist for re-usage, see *Runner types*.

Scenario - Type/class of measurement for example Ping, Pktgen, (Iperf, LmBench, ...)

SLA - Relates to what result boundary a test case must meet to pass. For example a latency limit, amount or ratio of lost packets and so on. Action based on SLA can be configured, either just to log (monitor) or to stop further testing (assert). The SLA criteria is set in the benchmark configuration file and evaluated by the runner.

3.2.3 Runner types

There exists several predefined runner types to choose between when designing a test scenario:

Arithmetic: Every test run arithmetically steps the specified input value(s) in the test scenario, adding a value to the previous input value. It is also possible to combine several input values for the same test case in different combinations.

Snippet of an Arithmetic runner configuration:

```
runner:
  type: Arithmetic
  iterators:
  -
    name: stride
    start: 64
    stop: 128
    step: 64
```

Duration: The test runs for a specific period of time before completed.

Snippet of a Duration runner configuration:

```
runner:
  type: Duration
  duration: 30
```

Sequence: The test changes a specified input value to the scenario. The input values to the sequence are specified in a list in the benchmark configuration file.

Snippet of a Sequence runner configuration:

```
runner:
  type: Sequence
  scenario_option_name: packetsize
  sequence:
  - 100
  - 200
  - 250
```

Iteration: Tests are run a specified number of times before completed.

Snippet of an Iteration runner configuration:

```
runner:
  type: Iteration
  iterations: 2
```

3.3 Use-Case View

Yardstick Use-Case View shows two kinds of users. One is the Tester who will do testing in cloud, the other is the User who is more concerned with test result and result analyses.

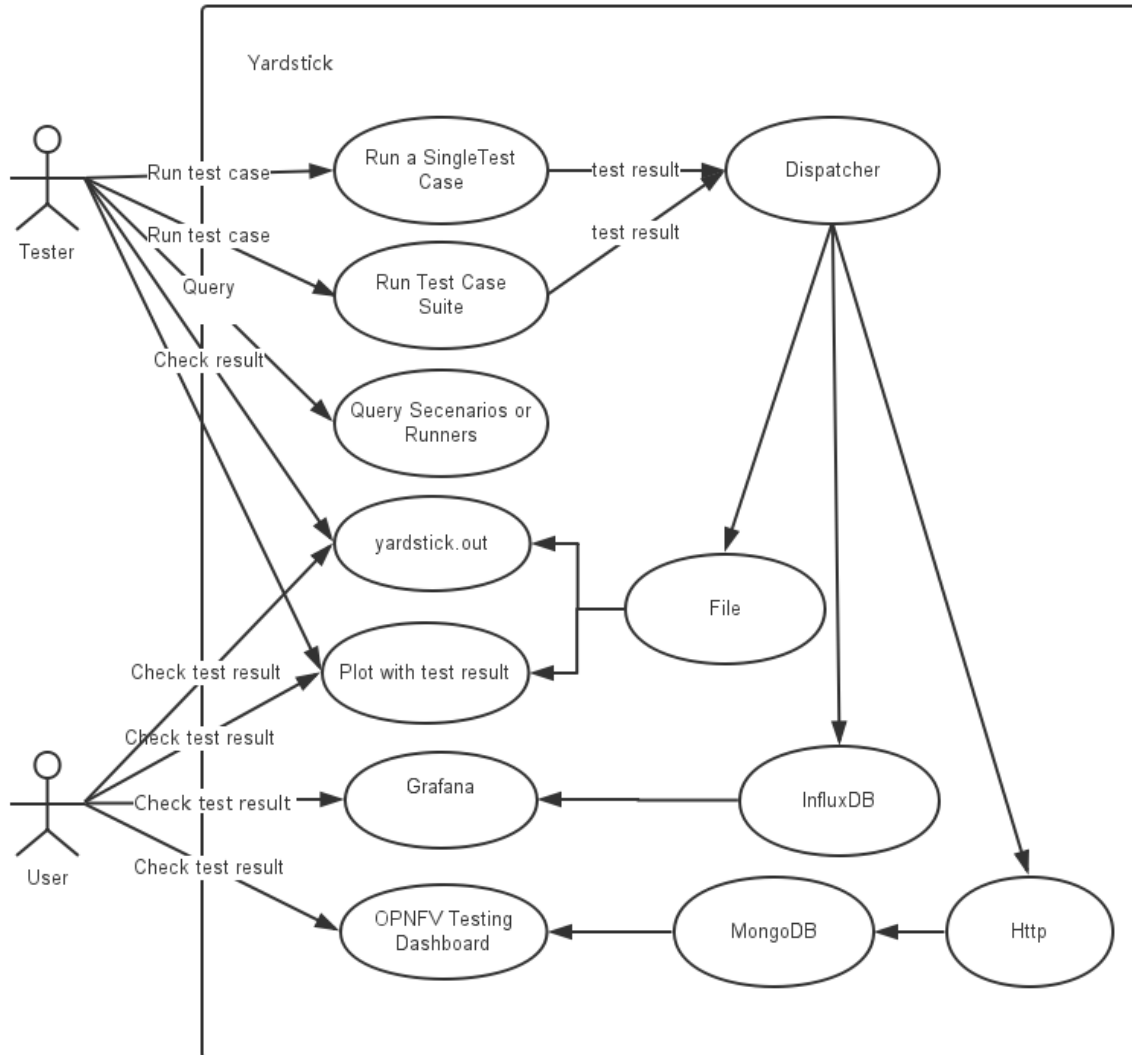
For testers, they will run a single test case or test case suite to verify infrastructure compliance or benchmark their own infrastructure performance. Test result will be stored by dispatcher module, three kinds of store method (file, influxdb and http) can be configured. The detail information of scenarios and runners can be queried with CLI by testers.

For users, they would check test result with four ways.

If dispatcher module is configured as file(default), there are two ways to check test result. One is to get result from yardstick.out (default path: /tmp/yardstick.out), the other is to get plot of test result, it will be shown if users execute command “yardstick-plot”.

If dispatcher module is configured as influxdb, users will check test result on Grafana which is most commonly used for visualizing time series data.

If dispatcher module is configured as http, users will check test result on OPNFV testing dashboard which use MongoDB as backend.



3.4 Logical View

Yardstick Logical View describes the most important classes, their organization, and the most important use-case realizations.

Main classes:

TaskCommands - “yardstick task” subcommand handler.

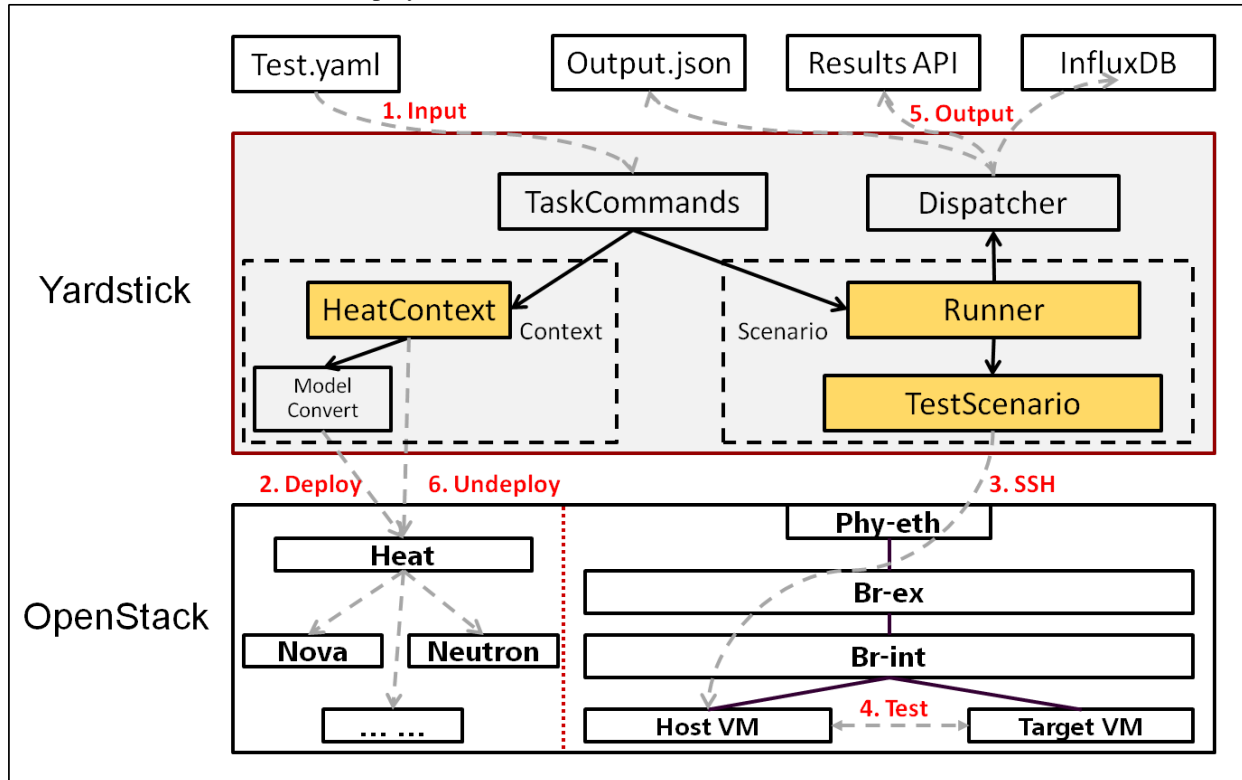
HeatContext - Do test yaml file context section model convert to HOT, deploy and undeploy Openstack heat stack.

Runner - Logic that determines how a test scenario is run and reported.

TestScenario - Type/class of measurement for example Ping, Pktgen, (Iperf, LmBench, ...)

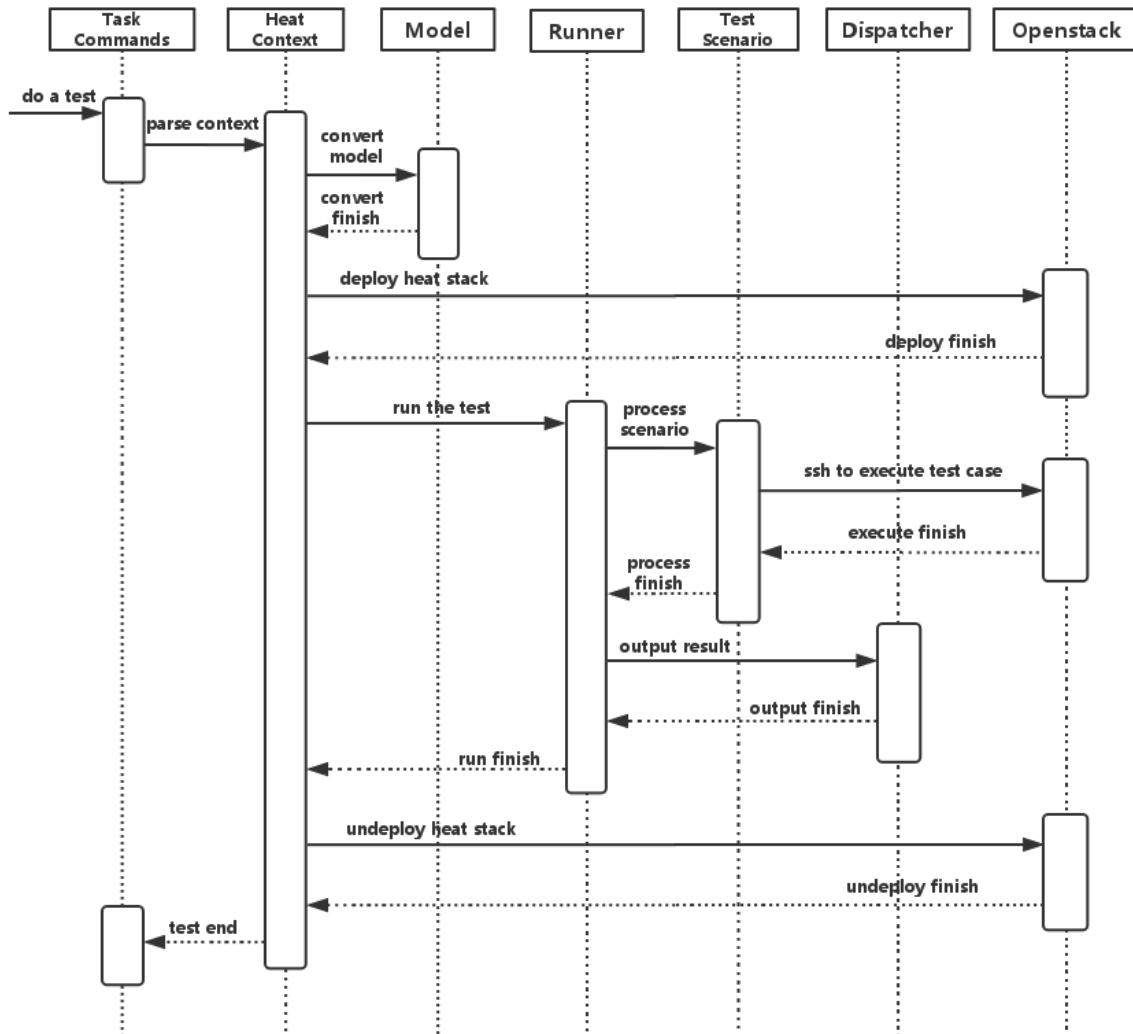
Dispatcher - Choose user defined way to store test results.

TaskCommands is the “yardstick task” subcommand’s main entry. It takes yaml file (e.g. test.yaml) as input, and uses HeatContext to convert the yaml file’s context section to HOT. After Openstack heat stack is deployed by HeatContext with the converted HOT, TaskCommands use Runner to run specified TestScenario. During first runner initialization, it will create output process. The output process use Dispatcher to push test results. The Runner will also create a process to execute TestScenario. And there is a multiprocessing queue between each runner process and output process, so the runner process can push the real-time test results to the storage media. TestScenario is commonly connected with VMs by using ssh. It sets up VMs and run test measurement scripts through the ssh tunnel. After all TestScenario is finished, TaskCommands will undeploy the heat stack. Then the whole test is finished.



3.5 Process View (Test execution flow)

Yardstick process view shows how yardstick runs a test case. Below is the sequence graph about the test execution flow using heat context, and each object represents one module in yardstick:



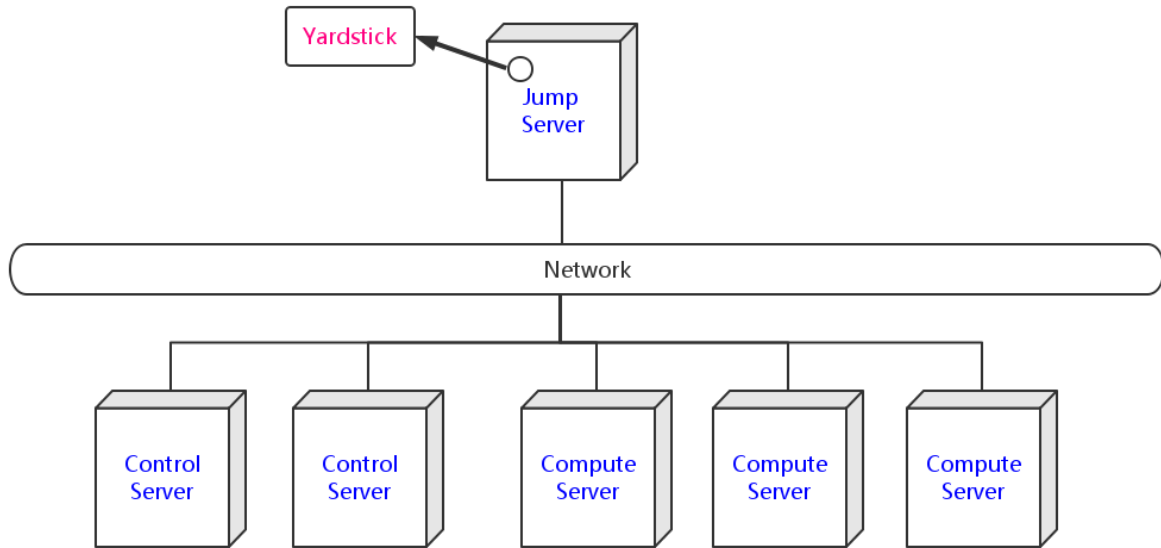
A user wants to do a test with yardstick. He can use the CLI to input the command to start a task. “TaskCommands” will receive the command and ask “HeatContext” to parse the context. “HeatContext” will then ask “Model” to convert the model. After the model is generated, “HeatContext” will inform “Openstack” to deploy the heat stack by heat template. After “Openstack” deploys the stack, “HeatContext” will inform “Runner” to run the specific test case.

Firstly, “Runner” would ask “TestScenario” to process the specific scenario. Then “TestScenario” will start to log on the openstack by ssh protocol and execute the test case on the specified VMs. After the script execution finishes, “TestScenario” will send a message to inform “Runner”. When the testing job is done, “Runner” will inform “Dispatcher” to output the test result via file, influxdb or http. After the result is output, “HeatContext” will call “Openstack” to undeploy the heat stack. Once the stack is undeployed, the whole test ends.

3.6 Deployment View

Yardstick deployment view shows how the yardstick tool can be deployed into the underlying platform. Generally, yardstick tool is installed on JumpServer(see *03-installation* for detail installation steps), and JumpServer is connected with other control/compute servers by networking. Based on this deployment, yardstick can run the test cases on these

hosts, and get the test result for better showing.



3.7 Yardstick Directory structure

yardstick/ - Yardstick main directory.

ci/ - Used for continuous integration of Yardstick at different PODs and with support for different installers.

docs/ - All documentation is stored here, such as configuration guides, user guides and Yardstick descriptions.

etc/ - Used for test cases requiring specific POD configurations.

samples/ - test case samples are stored here, most of all scenario and feature's samples are shown in this directory.

tests/ - Here both Yardstick internal tests (*functional/* and *unit/*) as well as the test cases run to verify the NFVI (*opnfv/*) are stored. Also configurations of what to run daily and weekly at the different PODs is located here.

tools/ - Currently contains tools to build image for VMs which are deployed by Heat. Currently contains how to build the yardstick-trusty-server image with the different tools that are needed from within the image.

vTC/ - Contains the files for running the virtual Traffic Classifier tests.

yardstick/ - Contains the internals of Yardstick: Runners, Scenario, Contexts, CLI parsing, keys, plotting tools, dispatcher and so on.

VIRTUAL TRAFFIC CLASSIFIER

4.1 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*.

4.2 Overview

The virtual Traffic Classifier (*VTC*) *VNF*, comprises of a Virtual Network Function Component (*VNFC*). The *VNFC* contains both the Traffic Inspection module, and the Traffic forwarding module, needed to run the *VNF*. The exploitation of Deep Packet Inspection (*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 Packet Based per Flow State (term:*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.

4.3 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. Type of Service (*ToS*) field modification.

4.4 Architecture

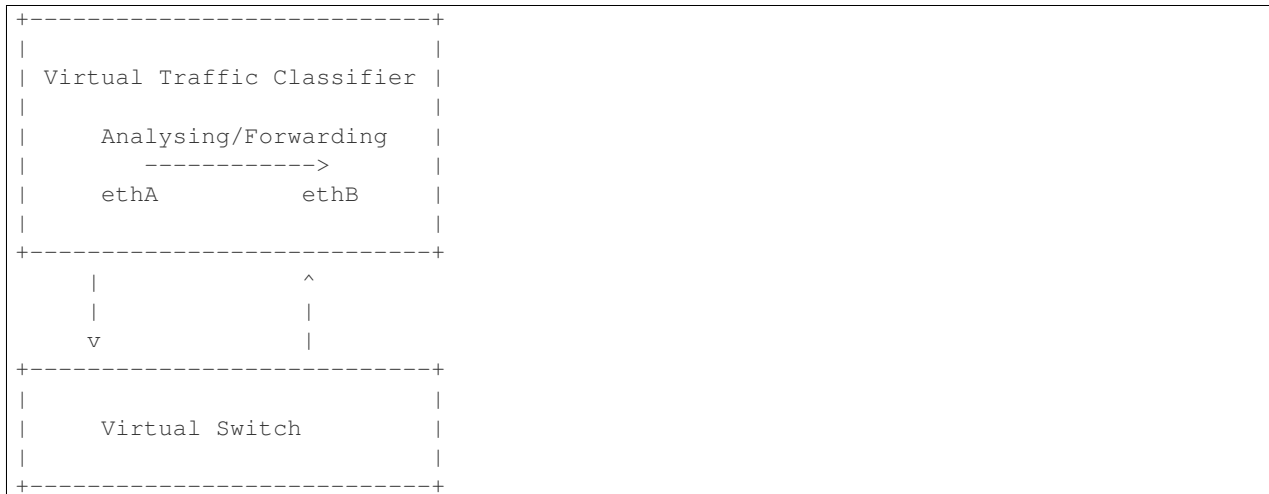
The Traffic Inspection module is the most computationally intensive component of the *VNF*. It implements filtering and packet matching algorithms in order to support the enhanced traffic forwarding capability of the *VNF*. The component supports a flow table (exploiting hashing algorithms for fast indexing 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 module 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*/Differentiated Services Code Point (*DSCP*) multimedia traffic for Quality of Service (*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.

4.5 Graphical Overview



4.6 Install

run the build.sh with root privileges

4.7 Run

sudo ./pfbridge -a eth1 -b eth2

4.8 Development Environment

Ubuntu 14.04

APEXLAKE INSTALLATION GUIDE

5.1 Abstract

ApexLake is a framework that provides automatic execution of experiments and related data collection to enable a user validate infrastructure from the perspective of a Virtual Network Function (*VNF*).

In the context of Yardstick, a virtual Traffic Classifier (*VTC*) network function is utilized.

5.1.1 Framework Hardware Dependencies

In order to run the framework there are some hardware related dependencies for ApexLake.

The framework needs to be installed on the same physical node where *DPDK-pktgen* is installed.

The installation requires the physical node hosting the packet generator must have 2 NICs which are *DPDK* compatible.

The 2 NICs will be connected to the switch where the OpenStack VM network is managed.

The switch used must support multicast traffic and *IGMP* snooping. Further details about the configuration are provided at the following [here](#).

The corresponding ports to which the cables are connected need to be configured as VLAN trunks using two of the VLAN IDs available for Neutron. Note the VLAN IDs used as they will be required in later configuration steps.

5.1.2 Framework Software Dependencies

Before starting the framework, a number of dependencies must first be installed. The following describes the set of instructions to be executed via the Linux shell in order to install and configure the required dependencies.

1. Install Dependencies.

To support the framework dependencies the following packages must be installed. The example provided is based on Ubuntu and needs to be executed in root mode.

```
apt-get install python-dev
apt-get install python-pip
apt-get install python-mock
apt-get install tcpreplay
apt-get install libpcap-dev
```

2. Source OpenStack opennc file.

```
source openrc
```

3. Configure Openstack Neutron

In order to support traffic generation and management by the virtual Traffic Classifier, the configuration of the port security driver extension is required for Neutron.

For further details please follow the following link: [PORTSEC](#) This step can be skipped in case the target OpenStack is Juno or Kilo release, but it is required to support Liberty. It is therefore required to indicate the release version in the configuration file located in `./yardstick/vTC/apexlake/apexlake.conf`

4. Create Two Networks based on VLANs in Neutron.

To enable network communications between the packet generator and the compute node, two networks must be created via Neutron and mapped to the VLAN IDs that were previously used in the configuration of the physical switch. The following shows the typical set of commands required to configure Neutron correctly. The physical switches need to be configured accordingly.

```
VLAN_1=2032
VLAN_2=2033
PHYSNET=physnet2
neutron net-create apexlake_inbound_network \
    --provider:network_type vlan \
    --provider:segmentation_id $VLAN_1 \
    --provider:physical_network $PHYSNET

neutron subnet-create apexlake_inbound_network \
    192.168.0.0/24 --name apexlake_inbound_subnet

neutron net-create apexlake_outbound_network \
    --provider:network_type vlan \
    --provider:segmentation_id $VLAN_2 \
    --provider:physical_network $PHYSNET

neutron subnet-create apexlake_outbound_network 192.168.1.0/24 \
    --name apexlake_outbound_subnet
```

5. Download Ubuntu Cloud Image and load it on Glance

The virtual Traffic Classifier is supported on top of Ubuntu 14.04 cloud image. The image can be downloaded on the local machine and loaded on Glance using the following commands:

```
wget cloud-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-disk1.img
glance image-create \
    --name ubuntu1404 \
    --is-public true \
    --disk-format qcow \
    --container-format bare \
    --file trusty-server-cloudimg-amd64-disk1.img
```

6. Configure the Test Cases

The VLAN tags must also be included in the test case Yardstick yaml file as parameters for the following test cases:

- [Yardstick Test Case Description TC006](#)
- [Yardstick Test Case Description TC007](#)
- [Yardstick Test Case Description TC020](#)
- [Yardstick Test Case Description TC021](#)

Install and Configure DPDK Pktgen

Execution of the framework is based on DPDK Pktgen. If DPDK Pktgen has not installed, it is necessary to download, install, compile and configure it. The user can create a directory and download the dpdk packet generator source code:

```
cd experimental_framework/libraries
mkdir dpdk_pktgen
git clone https://github.com/pktgen/Pktgen-DPDK.git
```

For instructions on the installation and configuration of DPDK and DPDK Pktgen please follow the official DPDK Pktgen README file. Once the installation is completed, it is necessary to load the DPDK kernel driver, as follow:

```
insmod uio
insmod DPDK_DIR/x86_64-native-linuxapp-gcc/kmod/igb_uio.ko
```

It is necessary to set the configuration file to support the desired Pktgen configuration. A description of the required configuration parameters and supporting examples is provided in the following:

```
[PacketGen]
packet_generator = dpdk_pktgen

# This is the directory where the packet generator is installed
# (if the user previously installed dpdk-pktgen,
# it is required to provide the director where it is installed).
pktgen_directory = /home/user/software/dpdk_pktgen/dpdk/examples/pktgen/

# This is the directory where DPDK is installed
dpdk_directory = /home/user/apexlake/experimental_framework/libraries/Pktgen-DPDK/dpdk/

# Name of the dpdk-pktgen program that starts the packet generator
program_name = app/app/x86_64-native-linuxapp-gcc/pktgen

# DPDK coremask (see DPDK-Pktgen readme)
coremask = 1f

# DPDK memory channels (see DPDK-Pktgen readme)
memory_channels = 3

# Name of the interface of the pktgen to be used to send traffic (vlan_sender)
name_if_1 = plp1

# Name of the interface of the pktgen to be used to receive traffic (vlan_receiver)
name_if_2 = plp2

# PCI bus address correspondent to if_1
bus_slot_nic_1 = 01:00.0

# PCI bus address correspondent to if_2
bus_slot_nic_2 = 01:00.1
```

To find the parameters related to names of the NICs and the addresses of the PCI buses the user may find it useful to run the *DPDK* tool `nic_bind` as follows:

```
DPDK_DIR/tools/dpdk_nic_bind.py --status
```

Lists the NICs available on the system, and shows the available drivers and bus addresses for each interface. Please make sure to select NICs which are *DPDK* compatible.

Installation and Configuration of smcroute

The user is required to install smcroute which is used by the framework to support multicast communications.

The following is the list of commands required to download and install smroute.

```
cd ~
git clone https://github.com/troglobit/smcroute.git
cd smcroute
git reset --hard c3f5c56
sed -i 's/aclocal-1.11/aclocal/g' ./autogen.sh
sed -i 's/automake-1.11/automake/g' ./autogen.sh
./autogen.sh
./configure
make
sudo make install
cd ..
```

It is required to do the reset to the specified commit ID. It is also requires the creation a configuration file using the following command:

```
SMCROUTE_NIC=(name of the nic)
```

where name of the nic is the name used previously for the variable “name_if_2”. For example:

```
SMCROUTE_NIC=p1p2
```

Then create the smcroute configuration file /etc/smcroute.conf

```
echo mgroup from $SMCROUTE_NIC group 224.192.16.1 > /etc/smcroute.conf
```

At the end of this procedure it will be necessary to perform the following actions to add the user to the sudoers:

```
adduser USERNAME sudo
echo "user ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers
```

Experiment using SR-IOV Configuration on the Compute Node

To enable *SR-IOV* interfaces on the physical NIC of the compute node, a compatible NIC is required. NIC configuration depends on model and vendor. After proper configuration to support *SR-IOV*, a proper configuration of OpenStack is required. For further information, please refer to the *SRIOV* configuration guide

5.1.3 Finalize installation the framework on the system

The installation of the framework on the system requires the setup of the project. After entering into the apexlake directory, it is sufficient to run the following command.

```
python setup.py install
```

Since some elements are copied into the /tmp directory (see configuration file) it could be necessary to repeat this step after a reboot of the host.

APEXLAKE API INTERFACE DEFINITION

6.1 Abstract

The API interface provided by the framework to enable the execution of test cases is defined as follows.

6.2 init

static init()

Initializes the Framework

Returns None

6.3 execute_framework

static execute_framework (test_cases,
iterations,
heat_template,
heat_template_parameters,
deployment_configuration,
openstack_credentials)

Executes the framework according the specified inputs

Parameters

- **test_cases**

Test cases to be run with the workload (dict() of dict())

Example: test_case = dict()

test_case['name'] = 'module.Class'

test_case['params'] = dict()

test_case['params']['throughput'] = '1'

test_case['params']['vlan_sender'] = '1000'

test_case['params']['vlan_receiver'] = '1001'

```
test_cases = [test_case]
```

- **iterations** Number of test cycles to be executed (int)
- **heat_template** (string) File name of the heat template corresponding to the workload to be deployed. It contains the parameters to be evaluated in the form of #parameter_name. (See heat_templates/vTC.yaml as example).
- **heat_template_parameters** (dict) Parameters to be provided as input to the heat template. See http://docs.openstack.org/developer/heat/template_guide/hot_guide.html section “Template input parameters” for further info.
- **deployment_configuration** (dict[string] = list(strings)) Dictionary of parameters representing the deployment configuration of the workload.

The key is a string corresponding to the name of the parameter, the value is a list of strings representing the value to be assumed by a specific param. The parameters are user defined: they have to correspond to the place holders (#parameter_name) specified in the heat template.

Returns dict() containing results

YARDSTICK INSTALLATION

7.1 Abstract

Yardstick currently supports installation on Ubuntu 14.04 or by using a Docker image. Detailed steps about installing Yardstick using both of these options can be found below.

To use Yardstick you should have access to an OpenStack environment, with at least Nova, Neutron, Glance, Keystone and Heat installed.

The steps needed to run Yardstick are:

1. Install Yardstick and create the test configuration .yaml file.
2. Build a guest image and load the image into the OpenStack environment.
3. Create a Neutron external network and load OpenStack environment variables.
4. Run the test case.

7.2 Installing Yardstick on Ubuntu 14.04

7.2.1 Installing Yardstick framework

Install dependencies:

```
sudo apt-get install python-virtualenv python-dev
sudo apt-get install libffi-dev libssl-dev git
```

Create a python virtual environment, source it and update setuptools:

```
virtualenv ~/yardstick_venv
source ~/yardstick_venv/bin/activate
easy_install -U setuptools
```

Download source code and install python dependencies:

```
git clone https://gerrit.opnfv.org/gerrit/yardstick
cd yardstick
python setup.py install
```

There is also a YouTube video, showing the above steps:

7.2.2 Installing extra tools

yardstick-plot

Yardstick has an internal plotting tool `yardstick-plot`, which can be installed using the following command:

```
python setup.py develop easy_install yardstick[plot]
```

7.2.3 Building a guest image

Yardstick has a tool for building an Ubuntu Cloud Server image containing all the required tools to run test cases supported by Yardstick. It is necessary to have `sudo` rights to use this tool.

This image can be built using the following command while in the directory where Yardstick is installed (`~/yardstick` if the framework is installed by following the commands above):

```
sudo ./tools/yardstick-img-modify tools/ubuntu-server-cloudimg-modify.sh
```

Warning: the script will create files by default in: `/tmp/workspace/yardstick` and the files will be owned by `root`!

The created image can be added to OpenStack using the `glance image-create` or via the OpenStack Dashboard.

Example command:

```
glance --os-image-api-version 1 image-create \  
--name yardstick-trusty-server --is-public true \  
--disk-format qcow2 --container-format bare \  
--file /tmp/workspace/yardstick/yardstick-trusty-server.img
```

7.3 Installing Yardstick using Docker

Yardstick has two Docker images, first one (**Yardstick-framework**) serves as a replacement for installing the Yardstick framework in a virtual environment (for example as done in *Installing Yardstick framework*), while the other image is mostly for CI purposes (**Yardstick-CI**).

7.3.1 Yardstick-framework image

Download the source code:

```
git clone https://gerrit.opnfv.org/gerrit/yardstick
```

Build the Docker image and tag it as *yardstick-framework*:

```
cd yardstick  
docker build -t yardstick-framework .
```

Run the Docker instance:

```
docker run --name yardstick_instance -i -t yardstick-framework
```

To build a guest image for Yardstick, see *Building a guest image*.

7.3.2 Yardstick-CI image

Pull the Yardstick-CI Docker image from Docker hub:

```
docker pull opnfv/yardstick-ci
```

Run the Docker image:

```
docker run \
  --privileged=true \
  --rm \
  -t \
  -e "INSTALLER_TYPE=${INSTALLER_TYPE}" \
  -e "INSTALLER_IP=${INSTALLER_IP}" \
  opnfv/yardstick-ci \
  run_benchmarks
```

Where `${INSTALLER_TYPE}` can be `fuel`, `foreman` or `compass` and `${INSTALLER_IP}` is the installer master node IP address (i.e. 10.20.0.2 is default for fuel).

Basic steps performed by the **Yardstick-CI** container:

1. clone yardstick and releng repos
2. setup OS credentials (releng scripts)
3. install yardstick and dependencies
4. build yardstick cloud image and upload it to glance
5. upload cirros-0.3.3 cloud image to glance
6. run yardstick test scenarios
7. cleanup

7.4 OpenStack parameters and credentials

7.4.1 Yardstick-flavor

Most of the sample test cases in Yardstick are using an OpenStack flavor called *yardstick-flavor* which deviates from the OpenStack standard `m1.tiny` flavor by the disk size - instead of 1GB it has 3GB. Other parameters are the same as in `m1.tiny`.

7.4.2 Environment variables

Before running Yardstick it is necessary to export OpenStack environment variables from the OpenStack *openrc* file (using the `source` command) and export the external network name `export EXTERNAL_NETWORK="external-network-name"`, the default name for the external network is `net04_ext`.

Credential environment variables in the *openrc* file have to include at least:

- `OS_AUTH_URL`
- `OS_USERNAME`
- `OS_PASSWORD`

- OS_TENANT_NAME

7.4.3 Yardstick default key pair

Yardstick uses a SSH key pair to connect to the guest image. This key pair can be found in the `resources/files` directory. To run the `ping-hot.yaml` test sample, this key pair needs to be imported to the OpenStack environment.

7.5 Examples and verifying the install

It is recommended to verify that Yardstick was installed successfully by executing some simple commands and test samples. Below is an example invocation of `yardstick help` command and `ping.py` test sample:

```
yardstick -h
yardstick task start samples/ping.yaml
```

Each testing tool supported by Yardstick has a sample configuration file. These configuration files can be found in the **samples** directory.

Example invocation of `yardstick-plot` tool:

```
yardstick-plot -i /tmp/yardstick.out -o /tmp/plots/
```

Default location for the output is `/tmp/yardstick.out`.

More info about the tool can be found by executing:

```
yardstick-plot -h
```

YARDSTICK TEST CASES

8.1 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*.

8.2 Generic NFVI Test Case Descriptions

8.2.1 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 and 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. |

8.2.2 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. |

8.2.3 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. |

8.2.4 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 and 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. |

8.2.5 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 and 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. |

8.2.6 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 <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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. |

8.2.7 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 | <p>File: opnfv_yardstick_tc011.yaml</p> <ul style="list-style-type: none"> 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 | <p>iperf3</p> <p>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.</p> <p>(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.)</p> |
| references | <p>iperf3</p> <p>ETSI-NFV-TST001</p> |
| applicability | <p>Test can be configured with different</p> <ul style="list-style-type: none"> 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 | <p>The test case image needs to be installed into Glance with iperf3 included in the image.</p> <p>No POD specific requirements have been identified.</p> |
| test sequence | description and expected result |
| step 1 | <p>The hosts are installed, as server and client. iperf3 is invoked and logs are produced and stored.</p> <p>Result: Logs are stored.</p> |
| test verdict | Test should not PASS if any jitter is above the optional SLA value, or if there is a test case execution problem. |

8.2.8 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 | <p>File: opnfv_yardstick_tc012.yaml</p> <ul style="list-style-type: none"> • 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 | <p>Lmbench</p> <p>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.</p> |
| references | <p>man-pages</p> <p>McVoy, Larry W., and Carl Staelin. "lmbench: Portable Tools for Performance Analysis." USENIX annual technical conference. 1996.</p> |
| applicability | <p>Test can be configured with different:</p> <ul style="list-style-type: none"> • memory sizes; • memory operations (such as rd, wr, rdwr, cp, frd, fwr, fcp, bzero, bcopy); • number of warmup iterations; • iterations and intervals. <p>There are default values for each above-mentioned option.</p> |
| pre-test conditions | <p>The test case image needs to be installed into Glance with Lmbench included in the image.</p> <p>No POD specific requirements have been identified.</p> |
| test sequence | description and expected result |
| step 1 | <p>The host is installed as client. Lmbench's bw_mem tool is invoked and logs are produced and stored.</p> <p>Result: logs are stored.</p> |
| test verdict | <p>Test fails if the measured memory bandwidth is below the SLA value or if there is a test case execution problem.</p> |

8.2.9 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. |

8.2.10 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 and 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. |

8.2.11 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 and 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. |

8.2.12 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 and 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 | <p>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.)</p> <p>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)</p> <p>mpstat (Mpstat is not always part of a Linux distribution, hence it needs to be installed. It is part of the Yardstick Glance image.</p> |
| references | <p>Ping and Mpstat man pages</p> <p>pktgen</p> <p>ETSI-NFV-TST001</p> |
| applicability | <p>Test can be configured with different packet sizes, amount of flows and test duration. Default values exist.</p> <p>SLA (optional): max_ppm: The number of packets per million packets sent that are acceptable to loose, not received.</p> |
| pre-test conditions | <p>The test case image needs to be installed into Glance with pktgen included in it.</p> <p>No POD specific requirements have been identified.</p> |
| test sequence | description and expected result |
| step 1 | <p>The hosts are installed, as server and client. pktgen is invoked and logs are produced and stored.</p> <p>Result: Logs are stored.</p> |
| test verdict | Fails only if SLA is not passed, or if there is a test case execution problem. |

8.3 OPNFV Feature Test Cases

8.3.1 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, neutron-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 scripts. 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 scripts. 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 running 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 maximum time (seconds) from the process being killed to recovered |
| test tool | Developed by the project. Please see folder: “yardstick/benchmark/scenarios/availability/ha_tools” |
| 8.3. OPNFV Feature Test Cases references | 41 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 scrips. 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” |

8.3.2 IPv6

Yardstick Test Case Description TC027

| | |
|---|---|
| IPv6 connectivity between nodes on the tenant network | |
| test case id | OPNFV_YARDSTICK_TC027_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 30 ms. ipv6 test case can be configured as three independent modules (setup, run, teardown). if you only want to setup ipv6 testing environment, do some tests as you want, "run_step" of task yaml file should be configured as "setup". if you want to setup and run ping6 testing automatically, "run_step" should be configured as "setup, run". and if you have had a environment which has been setup, you only want to verify the connectivity of ipv6 network, "run_step" should be "run". Of course, default is that three modules run sequentially. |
| 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 benchmark, 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. For Brahma Putra, a compass_os_nosdn_ha deploy scenario is need. more installer and more sdn deploy scenario will be supported soon |
| test sequence | description and expected result |
| step 1 | To setup IPV6 testing environment: 1. disable security group 2. create (ipv6, ipv4) router, network and subnet 3. create VRouter, VM1, VM2 |
| step 2 | To run ping6 to verify IPV6 connectivity : 1. ssh to VM1 2. Ping6 to ipv6 router from VM1 3. Get the result(RTT) and logs are stored |
| step 3 | To teardown IPV6 testing environment 1. delete VRouter, VM1, VM2 2. delete (ipv6, ipv4) router, network and subnet 3. enable security group |
| test verdict | Test should not PASS if any RTT is above the optional SLA value, or if there is a test case execution problem. |

8.3.3 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 guest 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. |

8.3.4 Parser

Yardstick Test Case Description TC040

| Verify Parser Yang-to-Tosca | |
|-----------------------------|--|
| test case id | OPNFV_YARDSTICK_TC040 Verify Parser Yang-to-Tosca |
| metric | <ol style="list-style-type: none"> 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. |

8.3.5 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 | <p>file: file: opnfv_yardstick_tc006.yaml</p> <p>packet_size: size of the packets to be used during the throughput calculation. Allowed values: [64, 128, 256, 512, 1024, 1280, 1518]</p> <p>vnic_type: type of VNIC to be used.</p> <p>Allowed values are:</p> <ul style="list-style-type: none"> • normal: for default OvS port configuration • direct: for SR-IOV port configuration <p>Default value: None</p> <p>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</p> <p>vlan_sender: vlan tag of the network on which the vTC will receive traffic (VLAN Network 1). Allowed values: range (1, 4096)</p> <p>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)</p> <p>default_net_name: neutron name of the default network that is used for access to the internet from the vTC (vNIC 1).</p> <p>default_subnet_name: subnet name for vNIC1 (information available through Neutron).</p> <p>vlan_net_1_name: Neutron Name for VLAN Network 1 (information available through Neutron).</p> <p>vlan_subnet_1_name: Subnet Neutron name for VLAN Network 1 (information available through Neutron).</p> <p>vlan_net_2_name: Neutron Name for VLAN Network 2 (information available through Neutron).</p> <p>vlan_subnet_2_name: Subnet Neutron name for VLAN Network 2 (information available through Neutron).</p> |
| 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. |
| 48 | The vNIC type and flavor MUST be specified by the user. Chapter 8. Yardstick Test Cases |
| 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 | <p>file: opnfv_yardstick_tc007.yaml</p> <p>packet_size: size of the packets to be used during the throughput calculation. Allowed values: [64, 128, 256, 512, 1024, 1280, 1518]</p> <p>vnic_type: type of VNIC to be used. Allowed values are:</p> <ul style="list-style-type: none"> • normal: for default OvS port configuration • direct: for SR-IOV port configuration <p>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</p> <p>num_of_neighbours: Number of noisy neighbours (VMs) to be instantiated during the experiment. Allowed values: range (1, 10)</p> <p>amount_of_ram: RAM to be used by each neighbor. Allowed values: ['250M', '1G', '2G', '3G', '4G', '5G', '6G', '7G', '8G', '9G', '10G'] Default value: 256M</p> <p>number_of_cores: Number of noisy neighbours (VMs) to be instantiated during the experiment. Allowed values: range (1, 10) Default value: 1</p> <p>vlan_sender: vlan tag of the network on which the vTC will receive traffic (VLAN Network 1). Allowed values: range (1, 4096)</p> <p>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)</p> <p>default_net_name: neutron name of the default network that is used for access to the internet from the vTC (vNIC 1).</p> <p>default_subnet_name: subnet name for vNIC1 (information available through Neutron).</p> <p>vlan_net_1_name: Neutron Name for VLAN Network 1 (information available through Neutron).</p> <p>vlan_subnet_1_name: Subnet Neutron name for VLAN Network 1 (information available through Neutron).</p> <p>vlan_net_2_name: Neutron Name for VLAN Network 2 (information available through Neutron).</p> <p>vlan_subnet_2_name: Subnet Neutron name for VLAN Network 2 (information available through Neutron).</p> |
| 50 | Chapter 8: Yardstick Test Cases |
| 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 | <p>file: opnfv_yardstick_tc020.yaml</p> <p>vnic_type: type of vNIC to be used.</p> <p>Allowed values are:</p> <ul style="list-style-type: none"> • normal: for default OvS port configuration • direct: for SR-IOV port configuration <p>Default value: None</p> <p>vtc_flavor: OpenStack flavor to be used for the vTC</p> <p>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</p> <p>Default value: None</p> <p>vlan_sender: vlan tag of the network on which the vTC will receive traffic (VLAN Network 1). Allowed values: range (1, 4096)</p> <p>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)</p> <p>default_net_name: neutron name of the default network that is used for access to the internet from the vTC (vNIC 1).</p> <p>default_subnet_name: subnet name for vNIC1 (information available through Neutron).</p> <p>vlan_net_1_name: Neutron Name for VLAN Network 1 (information available through Neutron).</p> <p>vlan_subnet_1_name: Subnet Neutron name for VLAN Network 1 (information available through Neutron).</p> <p>vlan_net_2_name: Neutron Name for VLAN Network 2 (information available through Neutron).</p> <p>vlan_subnet_2_name: Subnet Neutron name for VLAN Network 2 (information available through Neutron).</p> |
| 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 |
| 52 | <p>configuration parameters.</p> <p>Chapter 8. Yardstick Test Cases</p> <ul style="list-style-type: none"> • Multicast traffic MUST be enabled on the network. <p>The Data network switches need to be configured in order to manage multicast traffic. Installation and configuration of smcroute</p> |

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 | <p>file: opnfv_yardstick_tc021.yaml</p> <p>vnic_type: type of vNIC to be used.</p> <p>Allowed values are:</p> <ul style="list-style-type: none"> normal: for default OvS port configuration direct: for SR-IOV port configuration <p>Default value: None</p> <p>vtc_flavor: OpenStack flavor to be used for the vTC</p> <p>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</p> <p>Default value: None</p> <p>num_of_neighbours: Number of noisy neighbours (VMs) to be instantiated during the experiment. Allowed values: range (1, 10)</p> <p>amount_of_ram: RAM to be used by each neighbor.</p> <p>Allowed values: ['250M', '1G', '2G', '3G', '4G', '5G', '6G', '7G', '8G', '9G', '10G']</p> <p>Deault value: 256M</p> <p>number_of_cores: Number of noisy neighbours (VMs) to be instantiated during the experiment. Allowed values: range (1, 10) Default value: 1</p> <p>vlan_sender: vlan tag of the network on which the vTC will receive traffic (VLAN Network 1). Allowed values: range (1, 4096)</p> <p>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)</p> <p>default_net_name: neutron name of the default network that is used for access to the internet from the vTC (vNIC 1).</p> <p>default_subnet_name: subnet name for vNIC1 (information available through Neutron).</p> <p>vlan_net_1_name: Neutron Name for VLAN Network 1 (information available through Neutron).</p> <p>vlan_subnet_1_name: Subnet Neutron name for VLAN Network 1 (information available through Neutron).</p> <p>vlan_net_2_name: Neutron Name for VLAN Network 2 (information available through Neutron).</p> <p>vlan_subnet_2_name: Subnet Neutron name for VLAN Network 2 (information available through Neutron).</p> |
| test tool | <p>DPDK pktgen</p> <p>DPDK Pktgen is not part of a Linux distribution, hence it needs to be installed by the user.</p> |
| 54 references | <p>Chapter 8. Yardstick Test Cases</p> <p>DPDK Pktgen: DPDK Pktgen</p> <p>ETSI-NFV-TST001</p> <p>RFC 2544: rfc2544</p> |
| applicability | Test can be configured with different flavors. vNIC type |

8.4 Templates

8.4.1 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 performed, 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 several 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 |

8.4.2 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 packet size to compare the performance. The most elegant solution is then to turn the packet size name into a template variable:

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

schema: "yardstick:task:0.1"
scenarios:
-
  type: Ping
  options:
    packet_size: {{packet_size}}
  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 `{{packet_size}}` 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 '{"packet_size": "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 packet_size = packet_size 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
...

```


GLOSSARY

API Application Programming Interface
DPDK Data Plane Development Kit
DPI Deep Packet Inspection
DSCP Differentiated Services Code Point
IGMP Internet Group Management Protocol
IOPS Input/Output Operations Per Second
NFVI Network Function Virtualization Infrastructure
NIC Network Interface Controller
PBFS Packet Based per Flow State
QoS Quality of Service
SR-IOV Single Root IO Virtualization
SUT System Under Test
ToS Type of Service
VLAN Virtual LAN
VM Virtual Machine
VNF Virtual Network Function
VNFC Virtual Network Function Component
VTC Virtual Traffic Classifier

REFERENCES

10.1 OPNFV

- Parser wiki: <https://wiki.opnfv.org/parser>
- Pharos wiki: <https://wiki.opnfv.org/pharos>
- VTC: <https://wiki.opnfv.org/vtc>
- Yardstick CI: <https://build.opnfv.org/ci/view/yardstick/>
- Yardstick and ETSI TST001 presentation: https://wiki.opnfv.org/_media/opnfv_summit_-_bridging_opnfv_and_etsi.pdf
- Yardstick Project presentation: https://wiki.opnfv.org/_media/opnfv_summit_-_yardstick_project.pdf
- Yardstick wiki: <https://wiki.opnfv.org/yardstick>

10.2 References used in Test Cases

- cirros-image: <https://download.cirros-cloud.net>
- cyclictest: <https://rt.wiki.kernel.org/index.php/Cyclictest>
- DPDKpktgen: <https://github.com/Pktgen/Pktgen-DPDK/>
- DPDK supported NICs: <http://dpdk.org/doc/nics>
- fio: <http://www.bluestop.org/fio/HOWTO.txt>
- iperf3: <https://iperf.fr/>
- Lmbench man-pages: http://manpages.ubuntu.com/manpages/trusty/lat_mem_rd.8.html
- Memory bandwidth man-pages: http://manpages.ubuntu.com/manpages/trusty/bw_mem.8.html
- unixbench: <https://github.com/kdlucas/byte-unixbench/blob/master/UnixBench>
- mpstat man-pages: <http://manpages.ubuntu.com/manpages/trusty/man1/mpstat.1.html>
- pktgen: <https://www.kernel.org/doc/Documentation/networking/pktgen.txt>
- SR-IOV: <https://wiki.openstack.org/wiki/SR-IOV-Passthrough-For-Networking>

10.3 Research

- NCSR D: <http://www.demokritos.gr/?lang=en>
- T-NOVA: <http://www.t-nova.eu/>
- T-NOVA Results: <http://www.t-nova.eu/results/>

10.4 Standards

- ETSI NFV: <http://www.etsi.org/technologies-clusters/technologies/nfv>
- ETSI GS-NFV TST 001: https://docbox.etsi.org/ISG/NFV/Open/Drafts/TST001_-_Pre-deployment_Validation/
- RFC2544: <https://www.ietf.org/rfc/rfc2544.txt>

A

API, [59](#)

D

DPDK, [59](#)

DPI, [59](#)

DSCP, [59](#)

I

IGMP, [59](#)

IOPS, [59](#)

N

NFVI, [59](#)

NIC, [59](#)

P

PBFS, [59](#)

Q

QoS, [59](#)

S

SR-IOV, [59](#)

SUT, [59](#)

T

ToS, [59](#)

V

VLAN, [59](#)

VM, [59](#)

VNF, [59](#)

VNFC, [59](#)

VTC, [59](#)