OPNFV User Guide

Release arno.2015.1.0 (b6b86d4)

OPNFV

February 12, 2016

CHAPTER

ONE

ABSTRACT

OPNFV is a collaborative project aimed at providing a variety of virtualisation deployments intended to host applications serving the networking and carrier industry. This document provides guidance and instructions for using platform features designed to support these applications, made available in the Brahmaputra release of OPNFV.

This document is not intended to replace or replicate documentation from other open source projects such as OpenStack or OpenDaylight, rather highlight the features and capabilities delivered through the OPNFV project.

CHAPTER

TWO

OVERVIEW

OPNFV provides a variety of virtual infrastructure deployments designed to host virtualised network functions (VNFs). This guide intends to help users of the platform leverage the features and capabilities delivered by the OPNFV project.

OPNFV Continuous Integration builds, deploys and tests combinations of virtual infrastructure components in what are defined as scenarios. A scenario may include components such as OpenStack, OpenDaylight, OVS, KVM etc. where each scenario will include different source components or configurations. Scenarios are designed to enable specific features and capabilities in the platform that can be leveraged by the OPNFV user community.

2.1 OPNFV Scenarios

Each OPNFV scenario provides unique features and capabilities, it is important to ensure you have a scenario deployed on your infrastructure that provides the right capabilities for your needs before working through the user guide.

This user guide outlines how to work with key components and features in the platform, each feature description section will indicate the scenarios that provide the components and configurations required to use it.

Each scenario provides a set of platform capabilities and features that it supports. It is possible to identify which features are provided by reviewing the scenario name, however not all features and capabilities are discernible from the name itself.

2.1.1 Scenario Naming

In OPNFV, scenarios are identified by short scenario names. These names follow a scheme that identifies the key components and behaviours of the scenario, the rules for scenario naming are as follows:

os-[controller]-[feature]-[mode]-[option]

For example: *os-nosdn-kvm-noha* provides an OpenStack based deployment using neutron including the OPNFV enhanced KVM hypervisor.

The [feature] tag in the scenario name describes the main feature provided by the scenario. This scenario may also provide support for advanced fault management features which is not apparent in the scenario name. The following section describes the features available in each scenario.

2.1.2 Brahmaputra feature support matrix

The following table provides an overview of the available scenarios and supported features in the Brahmaputra release of OPNFV.

For details on which scenario's are best for you and how to install and configure them on your infrastructure the OPNFV Configuration guide provides a definitive reference.

The user guide will describe how to enable and utilise features and use cases implemented and tested on deployed OPNFV scenarios. For details of the use cases and tests that have been run you should check the validation procedures section of the features configuration guide. This will provide information about the specific use cases that have been validated and are working on your deployment.

2.2 General usage guidelines

The user guide for OPNFV features and capabilities provide step by step instructions for using features that have been configured according to the installation and configuration instructions.

This guide is structured in a manner that will provide usage instructions for each feature in its own section. Identify the feature capability you would like to leverage and read through that user guide section to understand the available usage. The combination of platform features, if available in a given scenario and not otherwise indicated, should function by following each features section. Dependencies between features will be highlighted in the user guide text.

You may wish to use the platform in a manner that the development team have not foreseen, or exercise capabilities not fully validated on the platform. If you experience issues leveraging the platform for the uses you have envisioned the OPNFV user mailing list provides a mechanism to establish a dialog with the community to help you overcome any issues identified.

It may be that you have identified a bug in the system, or that you are trying to execute a use case that has not yet ben implemented. In either case OPNFV is in essence a development project looking to ensure the required capabilities for our users are available.

CHAPTER

THREE

USING COMMON PLATFORM COMPONENTS

This section outlines basic usage principans and methods for some of the commonly deployed components of suported OPNFV scenario's in Brahmaputra. The subsections provide an outline of how these components are commonly used and how to address them in an OPNFV deployment. The components derive from autonomous upstream communities and where possible this guide will provide direction ot the relevant documentation made available by those communities to better help you navigate the OPNFV deployment.

3.1 Brahmaputra OpenStack User Guide

OpenStack is a cloud operating system developed and released by the OpenStack project. OpenStack is used in OPNFV for controlling pools of compute, storage, and networking resources in a Pharos compliant infrastructure.

OpenStack is used in Brahmaputra to manage tenants (known in OpenStack as projects), users, services, images, flavours, and quotas across the Pharos infrastructure. The OpenStack interface provides the primary interface for an operational Brahmaputra deployment and it is from the "horizon console" that an OPNFV user will perform the majority of administrative and operational activities on the deployment.

3.1.1 OpenStack references

The OpenStack user guide provides details and descriptions of how to configure and interact with the OpenStack deployment. This guide can be used by lab engineers and operators to tune the OpenStack deployment to your liking.

Once you have configured OpenStack to your purposes, or the Brahmaputra deployment meets your needs as deployed, an operator, or administrator, will find the best guidance for working with OpenStack in the OpenStack administration guide.

3.1.2 Connecting to the OpenStack instance

Once familiar with the basic of working with OpenStack you will want to connect to the OpenStack instance via the Horizon Console. The Horizon console provide a Web based GUI that will allow you operate the deployment. To do this you should open a browser on the JumpHost to the following address and enter the username and password:

http://{Controller-VIP}:80/index.html> username: admin password: admin

Other methods of interacting with and configuring OpenStack,, like the REST API and CLI are also available in the Brahmaputra deployment, see the OpenStack administration guide for more information on using those interfaces.

3.1.3 Common SDN components

3.2 OpenDaylight User Guide

OpenDaylight is an SDN controller platform developed and released by the OpenDaylight project. The OpenDaylight controller is installed and configured in OPNFV as the networking component of a variety of OPNFV VNFi scenarios using the neutron ODL device driver as an integration point toward OpenStack.

OpenDaylight runs within a JVM that is installed in OPNFV within a container and integrated with OpenStack. The OpenDaylight instance can be configured through the OpenStack Horizon interface, or accessed directly from the OPNFV Jumphost. The Brahmaputra release of OPNFV integrates the latest Lithium stable release or when deploying an SFC or SDNVPN scenario will integrate a Beryllium release version.

3.2.1 OpenDaylight references

For an overview of the OpenDaylight controller a good reference is the Getting Started Guide. For more detailed information about using the platform the OpenDaylight User Guide provides a good feature by feature reference.

It is important when working on your Brahmaputra deployment to be aware of the configured state of the OpenDaylight controller in the scenario you have deployed, installing an SFC scenario will for instance configure the OpenDaylight controller with the required SFC Karaf features in the OpenDaylight controller. Make sure you read the installation and configuration guide carefully to understand the state of the deployed system.

3.2.2 Connecting to the OpenDaylight instance

Once you are familiar with the OpenDaylight controller and it's configuration you will want to connect to the OpenDaylight instance from the Jumphost. To do this you should open a browser on the JumpHost to the following address and enter the username and password:

http://{Controller-VIP}:8181/index.html> username: admin password: admin

Other methods of interacting with and configuring the controller, like the REST API and CLI are also available in the Brahmaputra deployment, see the OpenDaylight User Guide for more information on using those interfaces.

It is important to be aware that when working directly on the OpenDaylight controller the OpenStack instance will not always be aware of the changes you are making to the networking controller. This may result in unrecoverable inconsistencies in your deployment.

3.3 ONOS User Guide

ONOS is an SDN controller platform developed and released by the ONOS project. The ONOS controller is installed and configured in OPNFV as the networking component of a variety of OPNFV NFVI scenarios.

ONOS runs within a JVM instance and is integrated with OpenStack via a Neutron ML2 plugin. The ONOS instance can be configured through the OpenStack Neutron interface, or through native ONOS tools, from the OPNFV jumphost. The Brahmaputra release of OPNFV integrates the latest ONOS 1.4 (EMU) release version.

3.3.1 ONOS references

For an overview of the ONOS controller, please see User Guide. For more detailed information about the EMU version of ONOS, documentation is available on the ONOS download page.

3.3.2 Connecting to the ONOS instance

Once you are familiar with the ONOS controller and its configuration you will want to connect to the ONOS instance from the Jumphost. To do this you should open a browser on the JumpHost to the following address and enter the username and password:

http://{Controller-VIP}:8282/index.html> username: karaf password: karaf

Other methods of interacting with and configuring the controller, like the REST API and CLI are also available in the Brahmaputra deployment, see the ONOS User Guide for more information on using those interfaces.

It is important to be aware that when working directly on the ONOS controller the OpenStack instance will not always be aware of the changes you are making to the networking controller. This may result in unrecoverable inconsistencies in your deployment.

If you have any questions or need further assistance, you may also direct your queries to *ONOSFW Forum <http://forum.onosfw.com>*

CHAPTER

FOUR

USING THE TEST FRAMEWORKS IN OPNFV

Testing is one of the key activities in OPNFV, validation can include component level testing, system testing, automated deployment validation and performance charecteristics testing.

The following sections outline how to use the test projects delivering automated test suites and frameworks in the in the Brahmaputra release of OPNFV.

4.1 Overview of the test suites

Functest is the OPNFV project primarily targeting function testing. In the Continuous Integration pipeline, it is launched after an OPNFV fresh installation to validate and verify the basic functions of the infrastructure.

The current list of test suites can be distributed in 3 main domains:

Domain	Test suite	Comments
VIM (Virtualised Infrastructure Manager)	vPing	NFV "Hello World" using SSH connection and floatting IP
	vPing_userdata	Ping using userdata and cloud-init mechanism
	Tempest	OpenStack reference test suite `[2]`_
	Rally bench	OpenStack testing tool benchmarking OpenStack modules `[3]`_
Controllers	OpenDaylight	Opendaylight Test suite
	ONOS	Test suite of ONOS L2 and L3 functions
	OpenContrail	
Features	vIMS	Example of a real VNF deployment to show the NFV capabilities of the platform. The IP Multimedia Subsytem is a typical Telco test case, referenced by ETSI. It provides a fully functional VoIP System
	Promise	Resource reservation and management project to identify NFV related requirements and realize resource reservation for future usage by capacity

```
|                 |                | management of resource pools regarding    |
|                 |                | compute, network and storage.             |
|                 +----------------+-------------------------------------------+
|                 | SDNVPN         |                                           |
+-----------------+----------------+-------------------------------------------+
```

Functest includes different test suites with several test cases within. Some of the tests are developed by Functest team members whereas others are integrated from upstream communities or other OPNFV projects. For example, Tempest is the OpenStack integration test suite and Functest is in charge of the selection, integration and automation of the tests that fit in OPNFV.

The Tempest suite has been customized but no new test cases have been created. Some OPNFV feature projects (e.g. SDNVPN) have written some Tempest tests cases and pushed upstream to be used by Functest.

The results produced by the tests run from CI are pushed and collected in a NoSQL database. The goal is to populate the database with results from different sources and scenarios and to show them on a Dashboard.

There is no real notion of Test domain or Test coverage. Basic components (VIM, controllers) are tested through their own suites. Feature projects also provide their own test suites with different ways of running their tests.

vIMS test case was integrated to demonstrate the capability to deploy a relatively complex NFV scenario on top of the OPNFV infrastructure.

Functest considers OPNFV as a black box. OPNFV, since the Brahmaputra release, offers lots of potential combinations:

- 3 controllers (OpenDayligh, ONOS, OpenContrail)
- 4 installers (Apex, Compass, Fuel, Joid)

Most of the tests are runnable on any combination, but some others might have restrictions imposed by the installers or the available deployed features.

4.2 Executing the functest suites

4.2.1 Manual testing

Once the Functest docker container is running and Functest environment ready (through */home/opnfv/repos/functest/docker/prepare_env.sh* script), the system is ready to run the tests.

The script *run_tests.sh* launches the test in an automated way. Although it is possible to execute the different tests manually, it is recommended to use the previous shell script which makes the call to the actual scripts with the appropriate parameters.

It is located in *$repos_dir/functest/docker* and it has several options:

```
./run_tests.sh -h
Script to trigger the tests automatically.

usage:
    bash run_tests.sh [-h|--help] [-r|--report] [-n|--no-clean] [-t|--test <test_name>]

where:
    -h|--help         show this help text
    -r|--report       push results to database (false by default)
    -n|--no-clean     do not clean up OpenStack resources after test run
    -s|--serial       run tests in one thread
    -t|--test         run specific set of tests
```

```
      <test_name>     one or more of the following separated by comma:
                         vping_ssh,vping_userdata,odl,rally,tempest,vims,onos,promise,ovno

examples:
    run_tests.sh
    run_tests.sh --test vping,odl
    run_tests.sh -t tempest,rally --no-clean
```

The *-r* option is used by the OPNFV Continuous Integration automation mechanisms in order to push the test results into the NoSQL results collection database. This database is read only for a regular user given that it needs special rights and special conditions to push data.

The *-t* option can be used to specify the list of a desired test to be launched, by default Functest will launch all the test suites in the following order: vPing, Tempest, vIMS, Rally.

A single or set of test may be launched at once using *-t <test_name>* specifying the test name or names separated by commas in the following list: *[vping,vping_userdata,odl,rally,tempest,vims,onos,promise]*.

The *-n* option is used for preserving all the possible OpenStack resources created by the tests after their execution.

Please note that Functest includes cleaning mechanism in order to remove all the VIM resources except what was present before running any test. The script *$repos_dir/functest/testcases/VIM/OpenStack/CI/libraries/generate_defaults.py* is called once by *prepare_env.sh* when setting up the Functest environment to snapshot all the OpenStack resources (images, networks, volumes, security groups, tenants, users) so that an eventual cleanup does not remove any of this defaults.

The *-s* option forces execution of test cases in a single thread. Currently this option affects Tempest test cases only and can be used e.g. for troubleshooting concurrency problems.

The script *$repos_dir/functest/testcases/VIM/OpenStack/CI/libraries/clean_openstack.py* is normally called after a test execution if the *-n* is not specified. It is in charge of cleaning the OpenStack resources that are not specified in the defaults file generated previously which is stored in */home/opnfv/functest/conf/os_defaults.yaml* in the docker container.

It is important to mention that if there are new OpenStack resources created manually after preparing the Functest environment, they will be removed if this flag is not specified in the *run_tests.sh* command. The reason to include this cleanup meachanism in Functest is because some test suites such as Tempest or Rally create a lot of resources (users, tenants, networks, volumes etc.) that are not always properly cleaned, so this cleaning function has been set to keep the system as clean as it was before a full Functest execution.

Within the Tempest test suite it is possible to define which test cases to execute by editing *test_list.txt* file before executing *run_tests.sh* script. This file is located in *$repos_dir/functest/testcases/VIM/OpenStack/CI/custom_tests/test_list.txt*

Although *run_tests.sh* provides an easy way to run any test, it is possible to do a direct call to the desired test script. For example:

```
python $repos_dir/functest/testcases/vPing/vPing.py -d
```

4.2.2 Automated testing

As mentioned in *[1]*, the *prepare-env.sh* and *run_test.sh* can be called within the container from Jenkins. There are 2 jobs that automate all the manual steps explained in the previous section. One job runs all the tests and the other one allows testing test suite by test suite specifying the test name. The user might use one or the other job to execute the desired test suites.

One of the most challenging task in the Brahmaputra release consists in dealing with lots of scenarios and installers. Thus, when the tests are automatically started from CI, a basic algorithm has been created in order to detect whether a given test is runnable or not on the given scenario. Some Functest test suites cannot be systematically run (e.g. ODL suite can not be run on an ONOS scenario).

CI provides some useful information passed to the container as environment variables:

- Installer (apex|compass|fuel|joid), stored in INSTALLER_TYPE
- Installer IP of the engine or VM running the actual deployment, stored in INSTALLER_IP
- The scenario [controller]-[feature]-[mode], stored in DEPLOY_SCENARIO with
  - controller = (odl|onos|oc|nosdn)
  - feature = (ovs(dpdk)|kvm)
  - mode = (ha|noha)

The constraints per test case are defined in the Functest configuration file */home/opnfv/functest/config/config_functest.yaml*:

```
test-dependencies:
   functest:
       vims:
           scenario: '(ocl)|(odl)|(nosdn)'
       vping:
       vping_userdata:
           scenario: '(ocl)|(odl)|(nosdn)'
       tempest:
       rally:
       odl:
           scenario: 'odl'
       onos:
           scenario: 'onos'
       ....
```

At the end of the Functest environment creation (prepare_env.sh see **'[1]'**_), a file */home/opnfv/functest/conf/testcase-list.txt* is created with the list of all the runnable tests. Functest considers the static constraints as regular expressions and compare them with the given scenario name. For instance, ODL suite can be run only on an scenario including 'odl' in its name.

The order of execution is also described in the Functest configuration file:

```
test_exec_priority:

   1: vping_ssh
   2: vping_userdata
   3: tempest
   4: odl
   5: onos
   6: ovno
   7: doctor
   8: promise
   9: odl-vpnservice
   10: bgpvpn
   11: openstack-neutron-bgpvpn-api-extension-tests
   12: vims
   13: rally
```

The tests are executed in the following order:

- Basic scenario (vPing_ssh, vPing_userdata, Tempest)
- Controller suites: ODL or ONOS or OpenContrail
- Feature projects (promise, vIMS)
- Rally (benchmark scenario)

As explained before, at the end of an automated execution, the OpenStack resources might be eventually removed.

CHAPTER

FIVE

GETTING STARTED WITH 'VSPERF'

VSPERF requires a traffic generators to run tests, automated traffic gen support in VSPERF includes:

- IXIA traffic generator (IxNetwork hardware) and a machine that runs the IXIA client software.
- Spirent traffic generator (TestCenter hardware chassis or TestCenter virtual in a VM) and a VM to run the Spirent Virtual Deployment Service image, formerly known as "Spirent LabServer".

If you want to use another traffic generator, please select the Dummy generator option as shown in Traffic generator instructions

To see the supported Operating Systems, vSwitches and system requirements, please follow the installation instructions to install.

Follow the Traffic generator instructions to install and configure a suitable traffic generator.

In order to run VSPERF, you will need to download DPDK and OVS. You can do this manually and build them in a preferred location, OR you could use vswitchperf/src. The vswitchperf/src directory contains makefiles that will allow you to clone and build the libraries that VSPERF depends on, such as DPDK and OVS. To clone and build simply:

```
$ cd src
$ make
```

VSPERF can be used with stock OVS (without DPDK support). When build is finished, the libraries are stored in src_vanilla directory.

The 'make' builds all options in src:

- Vanilla OVS
- OVS with vhost_user as the guest access method (with DPDK support)
- OVS with vhost_cuse s the guest access method (with DPDK support)

The vhost_user build will reside in src/ovs/ The vhost_cuse build will reside in vswitchperf/src_cuse The Vanilla OVS build will reside in vswitchperf/src_vanilla

To delete a src subdirectory and its contents to allow you to re-clone simply use:

```
$ make clobber
```

The `10_custom.conf` file is the configuration file that overrides default configurations in all the other configuration files in `./conf` The supplied `10_custom.conf` file **MUST** be modified, as it contains configuration items for which there are no reasonable default values.

The configuration items that can be added is not limited to the initial contents. Any configuration item mentioned in any .conf file in `./conf` directory can be added and that item will be overridden by the custom configuration value.

If your `10_custom.conf` doesn't reside in the `./conf` directory of if you want to use an alternative configuration file, the file can be passed to `vsperf` via the `--conf-file` argument.

```
$ ./vsperf --conf-file <path_to_custom_conf> ...
```

Note that configuration passed in via the environment (`--load-env`) or via another command line argument will override both the default and your custom configuration files. This "priority hierarchy" can be described like so (1 = max priority):

1. Command line arguments
2. Environment variables
3. Configuration file(s)

vsperf uses a VM called vloop_vnf for looping traffic in the PVP and PVVP deployment scenarios. The image can be downloaded from http://artifacts.opnfv.org/.

```
$ wget http://artifacts.opnfv.org/vswitchperf/vloop-vnf-ubuntu-14.04_20151216.qcow2
```

vloop_vnf forwards traffic through a VM using one of: * DPDK testpmd * Linux Bridge * l2fwd kernel Module.

Alternatively you can use your own QEMU image.

A Kernel Module that provides OSI Layer 2 Ipv4 termination or forwarding with support for Destination Network Address Translation (DNAT) for both the MAC and IP addresses. l2fwd can be found in <vswitchperf_dir>/src/l2fwd

Before running any tests make sure you have root permissions by adding the following line to /etc/sudoers:

```
username ALL=(ALL)       NOPASSWD: ALL
```

username in the example above should be replaced with a real username.

To list the available tests:

```
$ ./vsperf --list
```

To run a single test:

```
$ ./vsperf $TESTNAME
```

Where $TESTNAME is the name of the vsperf test you would like to run.

To run a group of tests, for example all tests with a name containing 'RFC2544':

```
$ ./vsperf --conf-file=<path_to_custom_conf>/10_custom.conf --tests="RFC2544"
```

To run all tests:

```
$ ./vsperf --conf-file=<path_to_custom_conf>/10_custom.conf
```

Some tests allow for configurable parameters, including test duration (in seconds) as well as packet sizes (in bytes).

```
$ ./vsperf --conf-file user_settings.py
    --tests RFC2544Tput
    --test-param "duration=10;pkt_sizes=128"
```

For all available options, check out the help dialog:

```
$ ./vsperf --help
```

1. If needed, recompile src for all OVS variants

```
$ cd src
$ make distclean
$ make
```

2. Update your "10_custom.conf" file to use the appropriate variables for Vanilla OVS:

```
VSWITCH = 'OvsVanilla'
VSWITCH_VANILLA_PHY_PORT_NAMES = ['$PORT1', '$PORT1']
```

Where $PORT1 and $PORT2 are the Linux interfaces you'd like to bind to the vswitch.

3. Run test:

```
$ ./vsperf --conf-file=<path_to_custom_conf>
```

Please note if you don't want to configure Vanilla OVS through the configuration file, you can pass it as a CLI argument; BUT you must set the ports.

```
$ ./vsperf --vswitch OvsVanilla
```

To run tests using vhost-user as guest access method:

1. Set VHOST_METHOD and VNF of your settings file to:

```
VHOST_METHOD='user'
VNF = 'QemuDpdkVhost'
```

2. If needed, recompile src for all OVS variants

```
$ cd src
$ make distclean
$ make
```

3. Run test:

```
$ ./vsperf --conf-file=<path_to_custom_conf>/10_custom.conf
```

To run tests using vhost-cuse as guest access method:

1. Set VHOST_METHOD and VNF of your settings file to:

```
VHOST_METHOD='cuse'
VNF = 'QemuDpdkVhostCuse'
```

2. If needed, recompile src for all OVS variants

```
$ cd src
$ make distclean
$ make
```

3. Run test:

```
$ ./vsperf --conf-file=<path_to_custom_conf>/10_custom.conf
```

To run tests using Vanilla OVS:

1. Set the following variables:

```
VSWITCH = 'OvsVanilla'
VNF = 'QemuVirtioNet'

VANILLA_TGEN_PORT1_IP = n.n.n.n
VANILLA_TGEN_PORT1_MAC = nn:nn:nn:nn:nn:nn

VANILLA_TGEN_PORT2_IP = n.n.n.n
VANILLA_TGEN_PORT2_MAC = nn:nn:nn:nn:nn:nn
```

```
VANILLA_BRIDGE_IP = n.n.n.n

or use --test-param

$ ./vsperf --conf-file=<path_to_custom_conf>/10_custom.conf
           --test-param "vanilla_tgen_tx_ip=n.n.n.n;
                         vanilla_tgen_tx_mac=nn:nn:nn:nn:nn:nn"
```

2. If needed, recompile src for all OVS variants

```
$ cd src
$ make distclean
$ make
```

3. Run test:

```
$ ./vsperf --conf-file<path_to_custom_conf>/10_custom.conf
```

To select loopback application, which will perform traffic forwarding inside VM, following configuration parameter should be configured:

```
GUEST_LOOPBACK = ['testpmd', 'testpmd']
```

or use –test-param

```
$ ./vsperf --conf-file=<path_to_custom_conf>/10_custom.conf
     --test-param "guest_loopback=testpmd"
```

Supported loopback applications are:

```
'testpmd'       - testpmd from dpdk will be built and used
'l2fwd'         - l2fwd module provided by Huawei will be built and used
'linux_bridge'  - linux bridge will be configured
'buildin'       - nothing will be configured by vsperf; VM image must
                  ensure traffic forwarding between its interfaces
```

Guest loopback application must be configured, otherwise traffic will not be forwarded by VM and testcases with PVP and PVVP deployments will fail. Guest loopback application is set to ‘testpmd’ by default.

To select application, which will perform packet forwarding, following configuration parameter should be configured:

```
    VSWITCH = 'none'
    PKTFWD = 'TestPMD'

    or use --vswitch and --fwdapp

    $ ./vsperf --conf-file user_settings.py
             --vswitch none
             --fwdapp TestPMD
```

Supported Packet Forwarding applications are:

```
    'testpmd'       - testpmd from dpdk
```

1. Update your “10_custom.conf” file to use the appropriate variables for selected Packet Forwarder:

```
    # testpmd configuration
    TESTPMD_ARGS = []
    # packet forwarding mode: io|mac|mac_retry|macswap|flowgen|rxonly|txonly|csum|icmpecho
    TESTPMD_FWD_MODE = 'csum'
    # checksum calculation layer: ip|udp|tcp|sctp|outer-ip
```

```
    TESTPMD_CSUM_LAYER = 'ip'
    # checksum calculation place: hw (hardware) | sw (software)
    TESTPMD_CSUM_CALC = 'sw'
    # recognize tunnel headers: on|off
    TESTPMD_CSUM_PARSE_TUNNEL = 'off'
```

2. Run test:

```
    $ ./vsperf --conf-file <path_to_settings_py>
```

VSPERF can be run in different modes. By default it will configure vSwitch, traffic generator and VNF. However it can be used just for configuration and execution of traffic generator. Another option is execution of all components except traffic generator itself.

Mode of operation is driven by configuration parameter -m or –mode

```
-m MODE, --mode MODE  vsperf mode of operation;
    Values:
        "normal" - execute vSwitch, VNF and traffic generator
        "trafficgen" - execute only traffic generator
        "trafficgen-off" - execute vSwitch and VNF
```

In case, that VSPERF is executed in “trafficgen” mode, then configuration of traffic generator should be configured through –test-param option. Supported CLI options useful for traffic generator configuration are:

```
'traffic_type'  - One of the supported traffic types. E.g. rfc2544,
                  back2back or continuous
                  Default value is "rfc2544".
'bidirectional' - Specifies if generated traffic will be full-duplex (true)
                  or half-duplex (false)
                  Default value is "false".
'iload'         - Defines desired percentage of frame rate used during
                  continuous stream tests.
                  Default value is 100.
'multistream'   - Defines number of flows simulated by traffic generator.
                  Value 0 disables MultiStream feature
                  Default value is 0.
'stream_type'   - Stream Type is an extension of the "MultiStream" feature.
                  If MultiStream is disabled, then Stream Type will be
                  ignored. Stream Type defines ISO OSI network layer used
                  for simulation of multiple streams.
                  Default value is "L4".
```

Example of execution of VSPERF in “trafficgen” mode:

```
$ ./vsperf -m trafficgen --trafficgen IxNet --conf-file vsperf.conf
    --test-params "traffic_type=continuous;bidirectional=True;iload=60"
```

Every developer participating in VSPERF project should run pylint before his python code is submitted for review. Project specific configuration for pylint is available at ‘pylint.rc’.

Example of manual pylint invocation:

```
$ pylint --rcfile ./pylintrc ./vsperf
```

If you encounter the following error: “before (last 100 chars): ‘-path=/dev/hugepages,share=on: unable to map backing store for hugepages: Cannot allocate memoryrnrn” with the PVP or PVVP deployment scenario, check the amount of hugepages on your system:

```
$ cat /proc/meminfo | grep HugePages
```

By default the vswitchd is launched with 1Gb of memory, to change this, modify –socket-mem parameter in conf/02_vswitch.conf to allocate an appropriate amount of memory:

```
VSWITCHD_DPDK_ARGS = ['-c', '0x4', '-n', '4', '--socket-mem 1024,0']
```

CHAPTER

SIX

USING BRAHMAPUTRA FEATURES

The following sections of the user guide provide feature specific usage guidelines and references. Providing users the necessary information to leveraging the features in the platform, some operation in this section may refer back to the guides in the general system usage section.

6.1 Copper capabilities and usage

This release focused on use of the OpenStack Congress service for managing configuration policy. See the Congress intro guide on readthedocs for information on the capabilities and usage of Congress.

6.2 Doctor capabilities and usage

6.2.1 Immediate Notification

Immediate notification can be used by creating 'event' type alarm via OpenStack Alarming (Aodh) API with relevant internal components support.

See, upstream spec document: http://specs.openstack.org/openstack/ceilometer-specs/specs/liberty/event-alarm-evaluator.html

You can find an example of consumer of this notification in doctor repository. It can be executed as follows:

```
git clone https://gerrit.opnfv.org/gerrit/doctor -b stable/brahmaputra
cd doctor/tests
CONSUMER_PORT=12346
python consumer.py "$CONSUMER_PORT" > consumer.log 2>&1 &
```

6.2.2 Consistent resource state awareness (Compute/host-down)

Resource state of compute host can be fixed according to an input from a monitor sitting out side of OpenStack Compute (Nova) by using force-down API.

See http://artifacts.opnfv.org/doctor/brahmaputra/docs/manuals/mark-host-down_manual.html for more detail.

CHAPTER

SEVEN

USING IPV6 FEATURE OF BRAHMAPUTRA RELEASE

This section provides the users with gap analysis regarding IPv6 feature requirements with OpenStack Kilo Official Release and Open Daylight Lithium Official Release. The gap analysis serves as feature specific user guides and references when as a user you may leverage the IPv6 feature in the platform and need to perform some IPv6 related operations.

This section provides users with IPv6 gap analysis regarding feature requirement with OpenStack Neutron in Kilo Official Release. The following table lists the use cases / feature requirements of VIM-agnostic IPv6 functionality, including infrastructure layer and VNF (VM) layer, and its gap analysis with OpenStack Neutron in Kilo Official Release.

Use Case / Requirement	Supported in Kilo Neutron	Notes
All topologies work in a multi-tenant environment	Yes	The IPv6 design is following the Neutron tenant networks model; dnsmasq is being used inside DHCP network namespaces, while radvd is being used inside Neutron routers namespaces to provide full isolation between tenants. Tenant isolation can be based on VLANs, GRE, or VXLAN encapsulation. In case of overlays, the transport network (and VTEPs) must be IPv4 based as of today.
IPv6 VM to VM only	Yes	It is possible to assign IPv6-only addresses to VMs. Both switching (within VMs on the same tenant network) as well as east/west routing (between different networks of the same tenant) are supported.
IPv6 external L2 VLAN directly attached to a VM	Yes	IPv6 provider network model; RA messages from upstream (external) router are forwarded into the VMs

Continued on next page

Table 7.1 – continued from previous page

Use Case / Requirement	Supported in Kilo Neutron	Notes
IPv6 subnet routed via L3 agent to an external IPv6 network 1. Both VLAN and overlay (e.g. GRE, VXLAN) subnet attached to VMs; 2. Must be able to support multiple L3 agents for a given external network to support scaling (neutron scheduler to assign vRouters to the L3 agents)	1. Yes 2. Yes	Configuration is enhanced in Kilo to allow easier setup of the upstream gateway, without the user forced to create an IPv6 subnet for the external network.
Ability for a NIC to support both IPv4 and IPv6 (dual stack) address. 1. VM with a single interface associated with a network, which is then associated with two subnets. 2. VM with two different interfaces associated with two different networks and two different subnets.	1. Yes 2. Yes	Dual-stack is supported in Neutron with the addition of `Multiple IPv6 Prefixes` Blueprint
Support IPv6 Address assignment modes. 1. SLAAC 2. DHCPv6 Stateless 3. DHCPv6 Stateful	1. Yes 2. Yes 3. Yes	
Ability to create a port on an IPv6 DHCPv6 Stateful subnet and assign a specific IPv6 address to the port and have it taken out of the DHCP address pool.	Yes	
Ability to create a port with fixed_ip for a SLAAC/DHCPv6-Stateless Subnet.	**No**	The following patch disables this operation: https://review.openstack.org/#/c/129144/
Support for private IPv6 to external IPv6 floating IP; Ability to specify floating IPs via Neutron API (REST and CLI) as well as via Horizon, including combination of IPv6/IPv4 and IPv4/IPv6 floating IPs if implemented.	**Rejected**	Blueprint proposed in upstream and got rejected. General expectation is to avoid NAT with IPv6 by assigning GUA to tenant VMs. See https://review.openstack.org/#/c/139731/ for discussion.

Continued on next page

Table 7.1 – continued from previous page

Use Case / Requirement	Supported in Kilo Neutron	Notes
Provide IPv6/IPv4 feature parity in support for pass-through capabilities (e.g., SR-IOV).	**To-Do**	The L3 configuration should be transparent for the SR-IOV implementation. SR-IOV networking support introduced in Juno based on the `sriovnicswitch` ML2 driver is expected to work with IPv4 and IPv6 enabled VMs. We need to verify if it works or not
Additional IPv6 extensions, for example: IPSEC, IPv6 Anycast, Multicast	**No**	It does not appear to be considered yet (lack of clear requirements)
VM access to the meta-data server to obtain user data, SSH keys, etc. using cloud-init with IPv6 only interfaces.	**No**	This is currently not supported. Config-drive or dual-stack IPv4 / IPv6 can be used as a workaround (so that the IPv4 network is used to obtain connectivity with the metadata service)
Full support for IPv6 matching (i.e., IPv6, ICMPv6, TCP, UDP) in security groups. Ability to control and manage all IPv6 security group capabilities via Neutron/Nova API (REST and CLI) as well as via Horizon.	Yes	
During network/subnet/router create, there should be an option to allow user to specify the type of address management they would like. This includes all options including those low priority if implemented (e.g., toggle on/off router and address prefix advertisements); It must be supported via Neutron API (REST and CLI) as well as via Horizon	Yes	Two new Subnet attributes were introduced to control IPv6 address assignment options: • `ipv6-ra-mode`: to determine who sends Router Advertisements; • `ipv6-address-mode`: to determine how VM obtains IPv6 address, default gateway, and/or optional information.
Security groups anti-spoofing: Prevent VM from using a source IPv6/MAC address which is not assigned to the VM	Yes	
Protect tenant and provider network from rough RAs	Yes	When using a tenant network, Neutron is going to automatically handle the filter rules to allow connectivity of RAs to the VMs only from the Neutron router port; with provider networks, users are required to specify the LLA of the upstream router during the subnet creation, or otherwise manually edit the security-groups rules to allow incoming traffic from this specific address.

Continued on next page

Table 7.1 – continued from previous page

Use Case / Requirement	Supported in Kilo Neutron	Notes
Support the ability to assign multiple IPv6 addresses to an interface; both for Neutron router interfaces and VM interfaces.	Yes	
Ability for a VM to support a mix of multiple IPv4 and IPv6 networks, including multiples of the same type.	Yes	
Support for IPv6 Prefix Delegation.	**Roadmap**	Some partial support is available in Liberty release
Distributed Virtual Routing (DVR) support for IPv6	**No**	Blueprint proposed upstream, pending discussion.
IPv6 First-Hop Security, IPv6 ND spoofing.	**Roadmap**	Supported in Liberty release
IPv6 support in Neutron Layer3 High Availability (keepalived+VRRP).	Yes	

This section provides users with IPv6 gap analysis regarding feature requirement with Open Daylight Lithium Official Release. The following table lists the use cases / feature requirements of VIM-agnostic IPv6 functionality, including infrastructure layer and VNF (VM) layer, and its gap analysis with Open Daylight Lithium Official Release.

Use Case / Requirement	Supported in ODL Lithium	Notes
REST API support for IPv6 subnet creation in ODL	Yes	Yes, it is possible to create IPv6 subnets in ODL using Neutron REST API. For a network which has both IPv4 and IPv6 subnets, ODL mechanism driver will send the port information which includes IPv4/v6 addresses to ODL Neutron northbound API. When port information is queried it displays IPv4 and IPv6 addresses. However, in Lithium release, ODL net-virt provider does not support IPv6 features (i.e., the actual functionality is missing and would be available only in the later releases of ODL).
IPv6 Router support in ODL 1. Communication between VMs on same compute node 2. Communication between VMs on different compute nodes (east-west) 3. External routing (north-south)	**No**	ODL net-virt provider in Lithium release only supports IPv4 Router. Support for IPv6 Router is planned in later releases using `Routing Manager`. In the meantime, if IPv6 Routing is necessary, we can use ODL for L2 connectivity and Neutron L3 agent for IPv4/v6 routing. **Note**: In Lithium SR3 release, we have the following issue, which is fixed upstream and back-ported to `stable/lithium` branch on December 15th, 2015.

Continued on next page

Table 7.2 – continued from previous page

Use Case / Requirement	Supported in ODL Lithium	Notes
IPAM: Support for IPv6 Address assignment modes. 1. SLAAC 2. DHCPv6 Stateless 3. DHCPv6 Stateful	**No**	Although it is possible to create different types of IPv6 subnets in ODL, ODL_L3 would have to implement the IPv6 Router that can send out Router Advertisements based on the IPv6 addressing mode. Router Advertisement is also necessary for VMs to configure the default route.
When using ODL for L2 forwarding/tunneling, is it compatible with IPv6.	Yes	
Full support for IPv6 matching (i.e., IPv6, ICMPv6, TCP, UDP) in security groups. Ability to control and manage all IPv6 security group capabilities via Neutron/Nova API (REST and CLI) as well as via Horizon.	**No**	Security Groups for IPv6 are currently not supported.
Shared Networks support	**No**	ODL currently assumes a single tenant to network mapping and does not support shared networks among tenants.
IPv6 external L2 VLAN directly attached to a VM.	**ToDo**	
ODL on an IPv6 only Infrastructure.	**ToDo**	Deploying OpenStack with ODL on an IPv6 only infrastructure where the API endpoints are all IPv6 addresses.

7.1 Promise capabilities and usage

Promise is a resource reservation and management project to identify NFV related requirements and realize resource reservation for future usage by capacity management of resource pools regarding compute, network and storage.

The following are the key features provided by this module:

- Capacity Management
- Reservation Management
- Allocation Management

The Brahmaputra implementation of Promise is built with the YangForge data modeling framework [1] , using a shim-layer on top of OpenStack to provide the Promise features. This approach requires communication between Consumers/Administrators and OpenStack to pass through the shim-layer. The shim-layer intercepts the message flow to manage the allocation requests based on existing reservations and available capacities in the providers. It also extracts information from the intercepted messages in order to update its internal databases. Furthermore, Promise provides additional intent-based APIs to allow a Consumer or Administrator to perform capacity management (i.e. add providers, update the capacity, and query the current capacity and utilization of a provider), reservation management (i.e. create, update, cancel, query reservations), and allocation management (i.e. create, destroy, query instances).

Detailed information about Promise use cases, features, interface specifications, work flows, and the underlying Promise YANG schema can be found in the Promise requirement document [2] .

[1] YangForge framework, http://github.com/opnfv/yangforge

[2] Promise requirement document, http://http://artifacts.opnfv.org/promise/docs/requirements/index.html

7.1.1 Promise usage

The yfc run command will load the primary application package from this repository along with any other dependency files/assets referenced within the YAML manifest and instantiate the opnfv-promise module and run REST/JSON interface by default listeningon port 5000.:

```
$ yfc run promise.yaml
```

You can also checkout the GIT repository (https://github.com/opnfv/promise/) or simply download the files into your local system and run the application.

7.1.2 Promise feature and API usage guidelines and examples

This section lists the Promise features and API implemented in OPNFV Brahmaputra.

Note 1: In contrast to ETSI NFV specifications and the detailed interface specification in Section 7, the Promise shim-layer implementation does not distinguish intent interfaces per resource type, i.e. the various capacity, reservations, etc. operations have different endpoints for each domain such as compute, storage, and network. The current shim-layer implementation does not separate the endpoints for performing the various operations.

Note 2: The listed parameters are optional unless explicitly marked as "mandatory".

Reservation management

The reservation management allows a Consumer to request reservations for resource capacity or specific resource elements. Reservations can be for now or a later time window. After the start time of a reservation has arrived, the Consumer can issue create server instance requests against the reserved capacity / elements. Note, a reservation will expire after a predefined *expiry* time in case no allocation referring to the reservation is requested.

The implemented workflow is well aligned with the described workflow in the Promise requirement document [1] (Clause 6.1) except for the "multi-provider" scenario as described in *(Multi-)provider management* .

create-reservation

This operation allows making a request to the reservation system to reserve resources. The Consumer can either request to reserve a certain capacity (*container*) or specific resource elements (*elements*), like a certain server instance.

The operation takes the following input parameters:

- start: start time of the requested reservation
- end: end time of the requested reservation
- container: request for reservation of capacity
  - instances: number of instances
  - cores: number of cores
  - ram: size of ram (in MB)
  - networks: number of networks
  - addresses: number of (public) IP addresses
  - ports: number of ports
  - routers: number of routers

  - subnets: number of subnets
  - gigabytes: size of storage (in GB)
  - volumes: number of volumes
  - snapshots: number of snapshots
- elements: reference to a list of 'pre-existing' resource elements that are required for fulfillment of the resource-usage-request
  - instance-identifier: identifier of a specific resource element
- zone: identifier of an Availability Zone

Promise will check the available capacity in the given time window and in case sufficient capacity exists to meet the reservation request, will mark those resources "reserved" in its reservation map.

update-reservation

This operation allows to update the reservation details for an existing reservation.

It can take the same input parameters as in *create-reservation* but in addition requires a mandatory reference to the *reservation-id* of the reservation that shall be updated.

cancel-reservation

This operation is used to cancel an existing reservation.

The operation takes the following input parameter:

- reservation-id (mandatory): identifier of the reservation to be canceled.

query-reservation

The operation queries the reservation system to return reservation(s) matching the specified query filter, e.g., reservations that are within a specified start/end time window.

The operation takes the following input parameters to narrow down the query results:

- zone: identifier of an Availability Zone
- without: excludes specified collection identifiers from the result
- elements:
  - some: query for ResourceCollection(s) that contain some or more of these element(s)
  - every: query for ResourceCollection(s) that contain all of these element(s)
- window: matches entries that are within the specified start/end time window
  - start: start time
  - end: end time
  - scope: if set to 'exclusive', only reservations with start AND end time within the time window are returned. Otherwise ('inclusive'), all reservation starting OR ending in the time windows are returned.
- show-utilization: boolean value that specifies whether to also return the resource utilization in the queried time window or not

subscribe-reservation-events / *notify-reservation-events*

Subscription to receive notifications about reservation-related events, e.g. a reservation is about to expire or a reservation is in conflict state due to a failure in the NFVI.

Note, this feature is not yet available in Brahmaputra release.

Allocation management

create-instance

This operation is used to create an instance of specified resource(s) for immediate use utilizing capacity from the pool. *Create-instance* requests can be issued against an existing reservation, but also allocations without a reference to an existing reservation are allowed. In case the allocation request specifies a reservation identifier, Promise checks if a reservation with that ID exists, the reservation start time has arrived (i.e. the reservation is 'active'), and the required capacity for the requested flavor is within the available capacity of the reservation. If those conditions are met, Promise creates a record for the allocation (VMState="INITIALIZED") and update its databases. If no *reservation_id* was provided in the allocation request, Promise checks whether the required capacity to meet the request can be provided from the available, non-reserved capacity. If yes, Promise creates a record for the allocation with an unique *instance-id* and update its databases. In any other case, Promise rejects the *create-instance* request.

In case the *create-instance* request is rejected, Promise responds with a "status=rejected" providing the reason of the rejection. This will help the Consumer to take appropriate actions, e.g., send an updated *create-instance* request. In case the *create-instance* request was accepted and a related allocation record has been created, the shim-layer issues a *createServer* request to the VIM Controller providing all information to create the server instance.

The operation takes the following input parameters:

- name (mandatory): Assigned name for the instance to be created
- image (mandatory): the image to be booted in the new instance
- flavor (mandatory): the flavor of the requested server instance
- networks: the list of network uuids of the requested server instance
- provider-id: identifier of the provider where the instance shall be created
- reservation-id: identifier of a resource reservation the *create-instance* is issued against

The Brahamputra implementation of Promise has the following limitations:

- All create server instance requests shall pass through the Promise shim-layer such that Promise can keep track of all allocation requests. This is necessary as in the current release the sychronization between the VIM Controller and Promise on the available capacity is not yet implemented.
- *Create-allocation* requests are limited to "simple" allocations, i.e., the current workflow only supports the Nova compute service and *create-allocation* requests are limited to creating one server instance at a time
- Prioritization of reservations and allocations is yet not implemented. Future version may allow certain policy-based conflict resolution where, e.g., new allocation request with high priority can "forcefully" terminate lower priority allocations.

destroy-instance

This operation request to destroy an existing server instance and release it back to the pool.

The operation takes the following input parameter:

- instance-id: identifier of the server instance to be destroyed

query-resource-collection

This operation allows to query for resource collection(s) that are within the specified start/end time window.

subscribe-allocation-events / *notify-allocation-events*

Subscription to receive notifications about allocation-related events, e.g. an allocation towards the VIM that did not pass the Promise shim-layer

Note, this feature is not yet available in Brahmaputra release.

Capacity management

The capacity management feature allows the Consumer or Administrator to do capacity planning, i.e. the capacity available to the reservation management can differ from the actual capacity in the registered provider(s). This feature can, e.g., be used to limit the available capacity for a given time window due to a planned downtime of some of the resources, or increase the capacity available to the reservation system in case of a plannes upgrade of the available capacity.

increase/decrease-capacity

This operations allows to increase/decrease the total capacity that is made available to the Promise reservation service between a specified window in time. It does NOT increase the actual capacity of a given resource provider, but is used for capacity management inside Promise.

This feature can be used in different ways, like

- Limit the capacity available to the reservation system to a value below 100% of the available capacity in the VIM, e.g., in order to leave "buffer" in the actual NFVI to be used outside the Promise reservation service.
- Inform the reservation system that, from a given time in the future, additional resources can be reserved, e.g., due to a planned upgrade of the available capacity of the provider.
- Similarily, the "decrease-capacity" can be used to reduce the consumable resources in a given time window, e.g., to prepare for a planned downtime of some of the resources.
- Expose multiple reservation service instances to different consumers sharing the same resource provider.

The operation takes the following input parameters:

- start: start time for the increased/decreased capacity
- end: end time for the increased/decreased capacity
- container: see *create-reservation*

Note, increase/decreasing the capacity in Promise is completely transparent to the VIM. As such, when increasing the virtual capacity in Promise (e.g. for a planned upgrade of the capacity), it is in the responsibility of the Consumer/Administrator to ensure sufficient resources in the VIM are available at the appropriate time, in order to prevent allocations against reservations to fail due to a lack of resources. Therefore, this operations should only be used carefully.

query-capacity

This operation is used to query the available capacity information of the specified resource collection. A filter attribute can be specified to narrow down the query results.

The current implementation supports the following filter criteria:

- time window: returns reservations matching the specified window
- window scope: if set to 'exclusive', only reservations with start AND end time within the time window are returned. Otherwise, all reservation starting OR ending in the time windows are returned.
- metric: query for one of the following capacity metrics:
  - 'total': resource pools
  - 'reserved': reserved resources
  - 'usage': resource allocations
  - 'available': remaining capacity, i.e. neither reserved nor allocated

subscribe-capacity-events / *notify-capacity-events*

These operations enable the Consumer to subscribe to receiving notifications about capacity-related events, e.g., increased/decreased capacity for a provider due to a failure or upgrade of a resource pool. In order to provide such notifications to its Consumers, Promise shim-layer has to subscribe itself to OpenStack Aodh to be notified from the VIM about any capacity related events.

Note, this feature is not yet available in Brahmaputra release.

(Multi-)provider management

This API towards OpenStack allows an Consumer/Administrator to add and remove resource providers to Promise. Note, Promise supports a multi-provider configuration, however, for Brahmaputra, multi-provider support is not yet fully supported.

add-provider

This operation is used to register a new resource provider into the Promise reservation system.

Note, for Brahmaputra, the add-provider operation should only be used to register one provider with the Promise shim-layer. Further note that currently only OpenStack is supported as a provider.

The operation takes the following input parameters:

- provider-type (mandatory) = 'openstack': select a specific resource provider type.
- endpoint (mandatory): target URL endpoint for the resource provider.
- username (mandatory)
- password (mandatory)
- region: specified region for the provider
- tenant
  - id
  - name

remove-provider

This operation removes a resource provider from the reservation system. Note, this feature is not yet available in Brahmaputra release.