NFVBENCH

Release Latest

Open Platform for NFV

Jul 30, 2021

CONTENTS

1 Introduction 3

2 OPNFV NFVbench Euphrates Design 5

3 OPNFV NFVbench Euphrates Design 11

4 NFVbench Release Notes 13

5 Developer Guide 17

6 Configuration Guide 19

7 NFVbench User Guide 21

- *Introduction*

CHAPTER

ONE

INTRODUCTION

NFVbench is a python application that is designed to run in a compact and portable format inside a container and on production pods. As such it only uses open sourec software with minimal hardware requirements (just a NIC card that is DPDK compatible). Traffic generation is handled by TRex on 2 physical ports (2x10G or higher) forming traffic loops up to VNF level and following a path that is common to all NFV applications: external source to top of rack switch(es) to compute node(s) to vswitch (if applicable) to VNF(s) and back.

Configuration of benchmarks is through a yaml configuraton file and command line arguments.

Results are available in different formats: - text output with tabular results - json result in file or in REST reply (most detailed)

Logging is available in a log file.

Benchmark results and logs can be optionally sent to one or more remote fluentd aggeregators using json format.

CHAPTER

TWO

OPNFV NFVBENCH EUPHRATES DESIGN

- *Introduction*
- *Configuration*
- *Orchestration*
- *Staging*
- *Traffic Generation*
- *Traffic Generator Results Analysis*

2.1 Introduction

NFVbench can be decomposed in the following components: - Configuration - Orchestration:

- Staging
- Traffic generation
- Results analysis

2.2 Configuration

This component is in charge of getting the configuration options from the user and consolidate them with the default configuration into a running configuration.

default configuration + user configuration options = running configuration

User configuration can come from: - CLI configuration shortcut arguments (e.g –frame-size) - CLI configuration file (–config [file]) - CLI configuration string (–config [string]) - REST request body - custom platform pluging

The precedence order for configuration is (from highest precedence to lowest precedence) - CLI configuration or REST configuration - custom platform plugin - default configuration

The custom platform plugin is an optional python class that can be used to override default configuration options with default platform options which can be either hardcoded or calculated at runtime from platform specific sources (such as platform deployment configuration files). A custom platform plugin class is a child of the parent class nfvbench.config_plugin.ConfigPlugin.

2.3 Orchestration

Once the configuration is settled, benchmark orchestration is managed by the ChainRunner class (nfvbench.chain_runner.ChainRunner). The chain runner will take care of orchestrating the staging, traffic generation and results analysis.

2.4 Staging

The staging component is in charge of staging the OpenStack resources that are used for the requested packet path. For example, for a PVP packet path, this module will create 2 Neutron networks and one VM instance connected to these 2 networks. Multi-chaining and VM placement is also handled by this module.

Main class: nfvbench.chaining.ChainManager

2.5 Traffic Generation

The traffic generation component is in charge of contrilling the TRex traffic generator using its python API. It includes tasks such as: - traffic check end to end to make sure the packet path is clear in both directions before starting a benchmark - programming the TRex traffic flows based on requested parameters - fixed rate control - NDR/PDR binary search

Main class: nfvbench.traffic_client.TrafficClient

2.6 Traffic Generator Results Analysis

At the end of a traffic generation session, this component collects the results from TRex and packages them in a format that is suitable for the various output formats (JSON, REST, file, fluentd). In the case of multi-chaining, it handles aggregation of results across chains.

Main class: nfvbench.stats_manager.StatsManager

2.7 Versioning

NFVbench uses semver compatible git tags such as “1.0.0”. These tags are also called project tags and applied at important commits on the master branch exclusively. Rules for the version numbers follow the semver 2.0 specification (https://semver.org). These git tags are applied indepently of the OPNFV release tags which are applied only on the stable release branches (e.g. “opnfv-5.0.0”).

In general it is recommeneded to always have a project git version tag associated to any OPNFV release tag content obtained from a sync from master.

NFVbench Docker containers will be versioned based on the NFVbench project tags.

2.8 Traffic Description

The general packet path model followed by NFVbench requires injecting traffic into an arbitrary number of service chains, where each service chain is identified by 2 edge networks (left and right). In the current multi-chaining model:

- all service chains can either share the same left and right edge networks or can have their own edge networks
- each port associated to the traffic generator is dedicated to send traffic to one side of the edge networks

If VLAN encapsulation is used, all traffic sent to a port will either have the same VLAN id (shared networks) or distinct VLAN ids (dedicated egde networks)

2.8.1 Basic Packet Description

The code to create the UDP packet is located in TRex.create_pkt() (nfvbench/traffic_gen/trex.py).

NFVbench always generates UDP packets (even when doing L2 forwarding). The final size of the frame containing each UDP packet will be based on the requested L2 frame size. When taking into account the minimum payload size requirements from the traffic generator for the latency streams, the minimum L2 frame size is 64 byte.

2.8.2 Flows Specification

Mac Addresses

The source MAC address is always the local port MAC address (for each port). The destination MAC address is based on the configuration and can be:

- the traffic generator peer port MAC address in the case of L2 loopback at the switch level or when using a loopback cable
- the dest MAC as specified by the configuration file (EXT chain no ARP)
- the dest MAC as discovered by ARP (EXT chain)
- the router MAC as discovered from Neutron API (PVPL3 chain)
- the VM MAC as dicovered from Neutron API (PVP, PVVP chains)

NFVbench does not currently range on the MAC addresses.

IP addresses

The source IP address is fixed per chain. The destination IP address is variable within a distinct range per chain.

UDP ports

The source and destination ports are fixed for all packets and can be set in the configuratoon file (default is 53).

Payload User Data

The length of the user data is based on the requested L2 frame size and takes into account the size of the L2 header - including the VLAN tag if applicable.

2.8.3 IMIX Support

In the case of IMIX, each direction is made of 4 streams: - 1 latency stream - 1 stream for each IMIX frame size

The IMIX ratio is encoded into the number of consecutive packets sent by each stream in turn.

2.8.4 Service Chains and Streams

A stream identifies one "stream" of packets with same characteristics such as rate and destination address. NFVbench will create 2 streams per service chain per direction:

- 1 latency stream set to 1000pps
- 1 main traffic stream set to the requested Tx rate less the latency stream rate (1000pps)

For example, a benchmark with 1 chain (fixed rate) will result in a total of 4 streams. A benchmark with 20 chains will results in a total of 80 streams (fixed rate, it is more with IMIX).

The overall flows are split equally between the number of chains by using the appropriate destination MAC address.

For example, in the case of 10 chains, 1M flows and fixed rate, there will be a total of 40 streams. Each of the 20 non-latency stream will generate packets corresponding to 50,000 flows (unique src/dest address tuples).

2.9 NDR/PDR Binary Search

The NDR/PDR binary search algorithm used by NFVbench is based on the algorithm used by the FD.io CSIT project, with some additional optimizations.

2.9.1 Algorithm Outline

The ServiceChain class (nfvbench/service_chain.py) is responsible for calculating the NDR/PDR or all frame sizes requested in the configuration. Calculation for 1 frame size is delegated to the TrafficClient class (nfvbench/traffic_client.py)

Call chain for calculating the NDR-PDR for a list of frame sizes:

- **ServiceChain.run()**
  - **ServiceChain._get_chain_results()**
    - **for every frame size:**
      - **ServiceChain.__get_result_per_frame_size()**

        TrafficClient.get_ndr_pdr()

        TrafficClient.__range_search() recursive binary search

The search range is delimited by a left and right rate (expressed as a % of line rate per direction). The search always start at line rate per port, e.g. in the case of 2x10Gbps, the first iteration will send 10Gbps of traffic on each port.

The load_epsilon configuration parameter defines the accuracy of the result as a % of line rate. The default value of 0.1 indicates for example that the measured NDR and PDR are within 0.1% of line rate of the actual NDR/PDR (e.g. 0.1% of 10Gbps is 10Mbps). It also determines how small the search range must be in the binary search. Smaller values of load_epsilon will result in more iterations and will take more time but may not always be beneficial if the absolute value falls below the precision level of the measurement. For example a value of 0.01% would translate to an absolute value of 1Mbps (for a 10Gbps port) or around 10kpps (at 64 byte size) which might be too fine grain.

The recursion narrows down the range by half and stops when:

- the range is smaller than the configured load_epsilon value
- or when the search hits 100% or 0% of line rate

2.9.2 Optimization

Binary search algorithms assume that the drop rate curve is monotonically increasing with the Tx rate. To save time, the algorithm used by NFVbench is capable of calculating the optimal Tx rate for an arbitrary list of target maximum drop rates in one pass instead of the usual 1 pass per target maximum drop rate. This saves time linearly to the number target drop rates. For example, a typical NDR/PDR search will have 2 target maximum drop rates:

- NDR = 0.001%
- PDR = 0.1%

The binary search will then start with a sorted list of 2 target drop rates: [0.1, 0.001]. The first part of the binary search will then focus on finding the optimal rate for the first target drop rate (0.1%). When found, the current target drop rate is removed from the list and iteration continues with the next target drop rate in the list but this time starting from the upper/lower range of the previous target drop rate, which saves significant time. The binary search continues until the target maximum drop rate list is empty.

2.9.3 Results Granularity

The binary search results contain per direction stats (forward and reverse). In the case of multi-chaining, results contain per chain stats. The current code only reports aggregated stats (forward + reverse for all chains) but could be enhanced to report per chain stats.

2.9.4 CPU Limitations

One particularity of using a software traffic generator is that the requested Tx rate may not always be met due to resource limitations (e.g. CPU is not fast enough to generate a very high load). The algorithm should take this into consideration:

- always monitor the actual Tx rate achieved as reported back by the traffic generator
- actual Tx rate is always <= requested Tx rate
- the measured drop rate should always be relative to the actual Tx rate
- if the actual Tx rate is < requested Tx rate and the measured drop rate is already within threshold (<NDR/PDR threshold) then the binary search must stop with proper warning because the actual NDR/PDR might probably be higher than the reported values

CHAPTER

THREE

OPNFV NFVBENCH EUPHRATES DESIGN

3.1 Building containers and VM images

NFVbench is delivered as Docker container which is built using the Dockerfile under the docker directory. This container includes the following parts:

- TRex traffic generator
- NFVbench orchestration
- NFVbench test VM (qcow2)

3.1.1 Versioning

These 3 parts are versioned independently and the Dockerfile will determine the combination of versions that are packaged in the container for the version associated to the Dockerfile.

The NFVbench version is controlled by the git tag that conforms to the semver version (e.g. "3.3.0"). This tag controls the version of the Dockerfile used for building the container.

The TRex version is controlled by the TREX_VER variable in Dockerfile (e.g. ENV TREX_VER "v2.56"). TRex is installed in container from https://github.com/cisco-system-traffic-generator/trex-core/releases

The Test VM version is controlled by the VM_IMAGE_VER variable in Dockerfile (e.g. ENV VM_IMAGE_VER "0.8"). The VM is extracted from google storage (http://artifacts.opnfv.org)

3.1.2 Updating the VM image

When the VM image is changed, its version must be increased in order to distinguish from previous image versions. The version strings to change are located in 2 files:

- docker/Dockerfile
- nfvbench/nfvbenchvm/dib/build-image.sh

3.1.3 Building and uploading the VM image

The VM image is built on gerrit verify when the image is not present in google storage. It is not uploaded yet on google storage.

The build + upload of the new VM image is done after the review is merged.

For details on how this is done, refer to ./jjb/nfvbench/nfvbench.yaml in the opnfv releng repository.

3.1.4 Building a new NFVbench container image

A new container image can be built and published to Dockerhub by CI/CD by applying a new semver tag to the nfvbench repository.

3.1.5 Workflow summary

NFVbench code has changed:

- commit with gerrit
- apply a new semver tag to trigger the container image build/publication

VM code has changed:

- update VM version in the 2 locations
- commit VM changes with gerrit to trigger VM build and publication to google storage
- IMPORTANT! wait for the VM image to be pushed to google storage before going to the next step (otherwise the container build will fail as it will not find the VM image)
- apply a new semver tag to trigger the container image build/publication

To increase the TRex version:

- change the Trex version in Dockerfile
- commit with gerrit
- apply a new semver tag to trigger the container image build/publication

CHAPTER

FOUR

NFVBENCH RELEASE NOTES

4.1 RELEASE NOTES

4.1.1 Release 3.6.2

NFVBENCH-152 Add service_mode method for debugging purpose NFVBENCH-150 Add support for VXLAN latency NFVBENCH-146 Add cache_size option NFVBENCH-151 Allocate hugepages on two NUMAs in nfvbenchvm NFVBENCH-149 Negative latency exception during NDR/PDR search NFVBENCH-148 Increase the waiting time based on # of instances

4.1.2 Release 3.5.1

- NFVBENCH-147 Incorrect URL used for admin check in credentials
- Release the validation check for VxLAN networks
- NFVBENCH-145 Config file not found. No explicit error
- NFVBENCH-144 Trex cannot take account NFVBench config (platform thread id 0)
- NFVBENCH-140 Retrieve High Dynamic Range latency histograms with TRex v2.59
- NFVBENCH-143 Trex cannot start due to invalid config (platform None)
- NFVBENCH-141 Fix Openstack user admin role check
- NFVBENCH-139 Fix master_thread_id and latency_thread_id property checking
- NFVBENCH-95 Add HdrHistogram encodes returned by TRex to JSON results
- NFVBENCH-138 Use yaml.safe_load() instead of unsafe yaml load
- NFVBENCH-137 NFVbench generates wrong L4 checksums for VxLAN traffic

4.1.3 Release 3.4.0

- Add L3 traffic management with Neutron routers

4.1.4 Release 3.3.0

Major release highlights:

- VxLAN support
- test VM can now have idle interfaces
- test VM can be launched with multiqueue enabled
- upgrade to TRex v2.56

4.1.5 Release 2.0

NFVbench will now follow its own project release numbering (x.y.z) which is independent of the OPNFV release numbering (opnfv-x.y.z)

Major release highlights:

- Dedicated edge networks for each chain
- Enhanced chain analysis
- Code refactoring and enhanced unit testing
- Miscellaneous enhancement

Dedicated edge networks for each chain

NFVbench 1.x only supported shared edge networks for all chains. For example, 20xPVP would create only 2 edge networks (left and right) shared by all chains. With NFVbench 2.0, chain networks are dedicated (unshared) by default with an option in the nfvbench configuration to share them. A 20xPVP run will create 2x20 networks instead.

Enhanced chain analysis

The new chain analysis improves at multiple levels:

- there is now one table for each direction (forward and reverse) that both read from left to right
- per-chain packet counters and latency
- all-chain aggregate packet counters and latency
- supports both shared and dedicated chain networks

Code refactoring and enhanced unit testing

The overall code structure is now better partitioned in the following functions:

- staging and resource discovery
- traffic generator
- stats collection

The staging algorithm was rewritten to be:

- a lot more robust to errors and to handle better resource reuse use cases. For example when a network with a matching name is discovered the new code will verify that the network is associated to the right VM instance

- a lot more strict when it comes to the inventory of MAC addresses. For example the association from each VM MAC to a chain index for each Trex port is handled in a much more strict manner.

Although not all code is unit tested, the most critical parts are unit tested with the use of the mock library. The resulting unit test code can run in isolation without needing a real system under test.

4.1.6 OPNFV Fraser Release

Over 30 Jira tickets have been addressed in this release (Jira NFVBENCH-55 to NFVBENCH-78)

The Fraser release adds the following new features:

- support for benchmarking non-OpenStack environments (with external setup and no OpenStack openrc file)
- PVVP packet path with SRIOV at the edge and vswitch between VMs
- support logging events and results through fluentd

Enhancements and main bug fixes:

- end to end connectivity for larger chain count is now much more accurate for large chain count - avoiding excessive drops
- use newer version of TRex (2.32)
- use newer version of testpmd DPDK
- NDR/PDR uses actual TX rate to calculate drops - resulting in more accurate results
- add pylint to unit testing
- add self sufficient and standalone unit testing (without actual testbed)

4.1.7 OPNFV Euphrates Release

This is the introductory release for NFVbench. In this release, NFVbench provides the following features/capabilities:

- standalone installation with a single Docker container integrating the open source TRex traffic generator
- can measure data plane performance for any NFVi full stack
- **can setup automatically service chains with the following packet paths:**
  - PVP (physical-VM-physical)
  - PVVP (physical-VM-VM-physical) intra-node and inter-node
- **can setup multiple service chains**
  - N * PVP
  - N * PVVP
- supports any external service chain (pre-set externally) that can do basic IPv4 routing
- **can measure**
  - drop rate and latency for any given fixed rate
  - NDR (No Drop Rate) and PDR (Partial Drop Rate) with configurable drop rates
- **traffic specification**
  - any fixed frame size or IMIX
  - uni or bidirectional traffic

    - any number of flows
    - vlan tagging can be enabled or disabled
- **user interface:**
    - CLI
    - REST+socketIO
- fully configurable runs with yaml-JSON configuration
- detailed results in JSON format
- summary tabular results
- can send logs and results to one or more fluentd aggregators (per configuration)

CHAPTER

FIVE

DEVELOPER GUIDE

CHAPTER

SIX

CONFIGURATION GUIDE

CHAPTER

SEVEN

NFVBENCH USER GUIDE

The NFVbench tool provides an automated way to measure the network performance for the most common data plane packet flows on any OpenStack system. It is designed to be easy to install and easy to use by non experts (no need to be an expert in traffic generators and data plane performance testing).

7.1 Table of Content

7.1.1 NFVbench: A Network Performance Benchmarking Tool for NFVi Full Stacks

The NFVbench tool provides an automated way to measure the network performance for the most common data plane packet flows on any NFVi system viewed as a black box (NFVi Full Stack). An NFVi full stack exposes the following interfaces: - an OpenStack API for those NFVi platforms based on OpenStack - an interface to send and receive packets on the data plane (typically through top of rack switches while simpler direct wiring to a looping device would also work)

The NFVi full stack can be any functional OpenStack system that provides the above interfaces. NFVbench can also be used without OpenStack on any networking device that can handle L2 forwarding or L3 routing.

NFVbench can be installed standalone (in the form of a single Docker container) and is fully functional without the need to install any other OPNFV tool.

It is designed to be easy to install and easy to use by non experts (no need to be an expert in traffic generators and data plane performance benchmarking). NFVbench integrates with the open source traffic generator TRex and provides the following benefits when compared to using a traffic generator directly:

- yaml configuration driven benchmark runs
- CLI or REST front end
- finds highest throughput based on drop rate requirement using an optimized binary search with very fast convergence time
- supports multi-chaining or dense VNF throughput measurement (e.g. find the throughput of a compute node running 20 loopback VNFs)
- detailed stats itemized per VNF chain in text or JSON format
- takes care of configuring packet flows and streams (often hard to use and specific to each gtraffic generator)
- takes care of bring up loopback VNFs/chains using Nova/Neutron/Glance OpenStack APIs
- saves you the hassle of searching what to measure, how to measure and how to interpret resuls

Data Plane Performance Measurement Features

NFVbench supports the following main measurement capabilities:

- **supports 2 measurement modes:**
  - *fixed rate* mode to generate traffic at a fixed rate for a fixed duration
  - NDR (No Drop Rate) and PDR (Partial Drop Rate) measurement mode
- configurable frame sizes (any list of fixed sizes or 'IMIX')
- built-in packet paths (PVP, PVVP)
- built-in loopback VNFs based on fast L2 or L3 forwarders running in VMs
- configurable number of flows and service chains
- configurable traffic direction (single or bi-directional)
- can support optional VLAN tagging (dot1q) or VxLAN overlays

NDR is the highest throughput achieved without dropping packets. PDR is the highest throughput achieved without dropping more than a pre-set limit (called PDR threshold or allowance, expressed in %).

Results of each run include the following data:

- Aggregated achieved bit rate throughput in bps
- Aggregated achieved packet rate in pps (or fps)
- Actual drop rate in %
- Latency in usec (min, max, average in the current version)

Built-in OpenStack support (optional)

NFVbench can optionally stage OpenStack resources to build 1 or more service chains using direct OpenStack APIs. Each service chain is composed of:

- 1 or 2 loopback VM instances per service chain
- 2 Neutron networks per loopback VM

OpenStack resources are staged before traffic is measured using OpenStack APIs (Nova and Neutron) then disposed after completion of measurements.

The loopback VM flavor to use can be configured in the NFVbench configuration file.

Note that NFVbench does not use OpenStack Heat nor any higher level service (VNFM or NFVO) to create the service chains because its main purpose is to measure the performance of the NFVi infrastructure which is mainly focused on L2 forwarding performance.

External Chains

NFVbench supports settings that involve externally staged packet paths with or without OpenStack:

- run benchmarks on existing service chains at the L3 level that are staged externally by any other tool (e.g. any VNF capable of L3 routing)
- run benchmarks on existing L2 chains that are configured externally (e.g. pure L2 forwarder such as DPDK testpmd)

Direct L2 Loopback (Switch or wire loopback)

NFVbench supports benchmarking of pure L2 loopbacks

- Switch level loopback
- Port to port wire loopback

In this mode, NFVbench will send packets from each port to the other port (the destination MAC address is set to the other port MAC address). This can be useful for example to verify that the connectivity to the switch is working properly.

Such a test can be quickly run using the CLI `--l2-loopback` *option*.

For a typical test, packets will be VLAN tagged with the same ID on both ports. However, multiple L2 vlan tagged service chains are also allowed, which permits testing various configurations and the behavior of the bench itself.

Traffic Generation

NFVbench currently integrates with the open source TRex traffic generator:

- TRex (pre-built into the NFVbench container)

Supported Packet Paths

Packet paths describe where packets are flowing in the NFVi platform. The most commonly used paths are identified by 3 or 4 letter abbreviations. A packet path can generally describe the flow of packets associated to one or more service chains, with each service chain composed of 1 or more VNFs.

The following packet paths are currently supported by NFVbench:

- PVP (Physical interface to VM to Physical interface)
- PVVP (Physical interface to VM to VM to Physical interface)
- N*PVP (N concurrent PVP packet paths)
- N*PVVP (N concurrent PVVP packet paths)

The traffic is made of 1 or more flows of L3 frames (UDP packets) with different payload sizes. Each flow is identified by a unique source and destination MAC/IP tuple.

Loopback VM

NFVbench provides a loopback VM image that runs CentOS with 2 pre-installed forwarders:

- DPDK testpmd configured to do L2 cross connect between 2 virtual interfaces
- FD.io VPP configured to perform L3 routing between 2 virtual interfaces

Frames are just forwarded from one interface to the other. In the case of testpmd, the source and destination MAC are rewritten, which corresponds to the mac forwarding mode (–forward-mode=mac). In the case of VPP, VPP will act as a real L3 router, and the packets are routed from one port to the other using static routes.

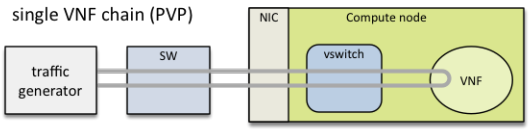
Which forwarder and what Nova flavor to use can be selected in the NFVbench configuration. Be default the DPDK testpmd forwarder is used with 2 vCPU per VM. The configuration of these forwarders (such as MAC rewrite configuration or static route configuration) is managed by NFVbench.

Importance of Dense VNF Measurement

Production deployments of NFVi stacks can require to run a large number of VMs per compute node in order to fully utilize all the hardware resources available in each of these compute nodes. Given that optimization of a compute node can be very different based on the number of VMs, it is therefore critical to do performance benchmarking at scale. NFVbench has been the first benchmarking tool to recognize this and to provide dense VNF dataplane benchmarking by staging multipe chains using OpenStack and configuring the traffic generator to split the traffic across all configured chains. This kind of measurement is very time consuming to do directly with traffic generators as it requires understanding how traffic is shaped in order to cover all chains in a balanced way.

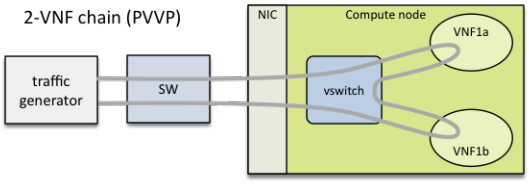
PVP Packet Path

This packet path represents a single service chain with 1 loopback VNF and 2 Neutron networks:



PVVP Packet Path

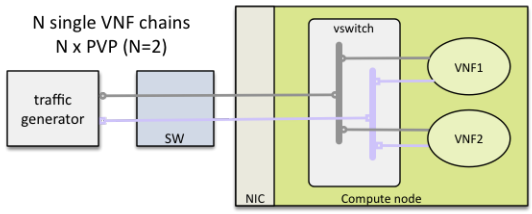
This packet path represents a single service chain with 2 loopback VNFs in sequence and 3 Neutron networks. The 2 VNFs will only run on the same compute node (PVVP intra-node):



Dense VNF or Multi-Chaining (N*PVP or N*PVVP)

Multiple service chains can be setup by NFVbench without any limit on the concurrency (other than limits imposed by available resources on compute nodes). In the case of multiple service chains, NFVbench will instruct the traffic generator to use multiple L3 packet streams (frames directed to each path will have a unique destination MAC address).

Example of multi-chaining with 2 concurrent PVP service chains:



This innovative feature will allow to measure easily the performance of a fully loaded compute node running multiple service chains.

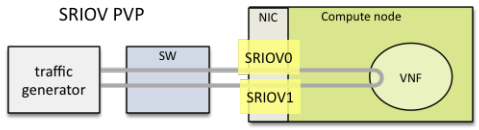
Multi-chaining is currently limited to 1 compute node (VMs run on the same compute node). The 2 edge interfaces for all service chains can either share the same 2 networks or can use dedicated networks (based on a configuration option). The total traffic will be split equally across all chains.

SR-IOV

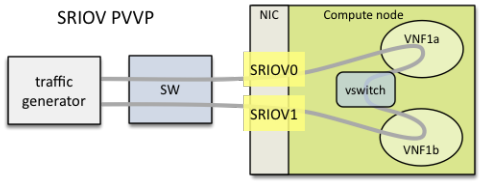
By default, service chains will be based on virtual switch interfaces.

NFVbench provides an option to select SR-IOV based virtual interfaces instead (thus bypassing any virtual switch) for those OpenStack system that include and support SR-IOV capable NICs on compute nodes.

The PVP packet path will bypass the virtual switch completely when the SR-IOV option is selected:

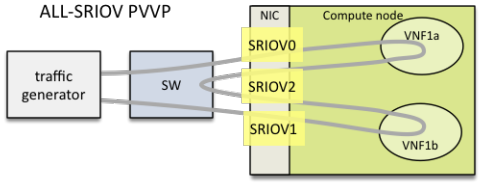


The PVVP packet path will use SR-IOV for the left and right networks and the virtual switch for the middle network by default:



This packet path is a good way to approximate VM to VM (V2V) performance (middle network) given the high efficiency of the left and right networks. The V2V throughput will likely be very close to the PVVP throughput while its latency will be very close to the difference between the SR-IOV PVVP latency and the SR-IOV PVP latency.

It is possible to also force the middle network to use SR-IOV (in this version, the middle network is limited to use the same SR-IOV phys net):



Other Misc Packet Paths

P2P (Physical interface to Physical interface - no VM) can be supported using the external chain/L2 forwarding mode.

V2V (VM to VM) is not supported but PVVP provides a more complete (and more realistic) alternative.

PVP chain with L3 routers in the path can be supported using PVP chain with L3 forwarding mode (l3_router option). See PVP L3 Router Internal Chain section for more details.

Supported Neutron Network Plugins and vswitches

Any Virtual Switch, Any Encapsulation

NFVbench is agnostic of the virtual switch implementation and has been tested with the following virtual switches:

- ML2/VPP/VLAN (networking-vpp)
- OVS/VLAN and OVS-DPDK/VLAN
- ML2/ODL/VPP (OPNFV Fast Data Stack)

7.1.2 Limitations

VxLAN: latency measurement is not available in the current VxLAN release PVVP Inter-node (where the 2 VMs are running on diofferent compute nodes) is no longer supported

7.1.3 Installation and Quick Start Guides

Requirements for running NFVbench

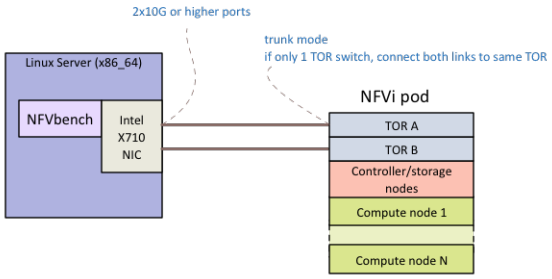
Hardware Requirements

To run NFVbench you need the following hardware: - a Linux server - a DPDK compatible NIC with at least 2 ports (preferably 10Gbps or higher) - 2 ethernet cables between the NIC and the OpenStack pod under test (usually through a top of rack switch)

The DPDK-compliant NIC must be one supported by the TRex traffic generator (such as Intel X710, refer to the Trex Installation Guide for a complete list of supported NIC)

To run the TRex traffic generator (that is bundled with NFVbench) you will need to wire 2 physical interfaces of the NIC to the

- if you have only 1 TOR, wire both interfaces to that same TOR
- 1 interface to each TOR if you have 2 TORs and want to use bonded links to your compute nodes



Switch Configuration

The 2 corresponding ports on the switch(es) facing the Trex ports on the Linux server should be configured in trunk mode (NFVbench will instruct TRex to insert the appropriate vlan tag).

Using a TOR switch is more representative of a real deployment and allows to measure packet flows on any compute node in the rack without rewiring and includes the overhead of the TOR switch.

Although not the primary targeted use case, NFVbench could also support the direct wiring of the traffic generator to a compute node without a switch.

Software Requirements

You need Docker to be installed on the Linux server.

TRex uses the DPDK interface to interact with the DPDK compatible NIC for sending and receiving frames. The Linux server will need to be configured properly to enable DPDK.

DPDK requires a uio (User space I/O) or vfio (Virtual Function I/O) kernel module to be installed on the host to work. There are 2 main uio kernel modules implementations (igb_uio and uio_pci_generic) and one vfio kernel module implementation.

To check if a uio or vfio is already loaded on the host:

```
lsmod | grep -e igb_uio -e uio_pci_generic -e vfio
```

If missing, it is necessary to install a uio/vfio kernel module on the host server:

- find a suitable kernel module for your host server (any uio or vfio kernel module built with the same Linux kernel version should work)
- load it using the modprobe and insmod commands

Example of installation of the igb_uio kernel module:

```
modprobe uio
insmod ./igb_uio.ko
```

Finally, the correct iommu options and huge pages to be configured on the Linux server on the boot command line:

- enable intel_iommu and iommu pass through: "intel_iommu=on iommu=pt"
- for Trex, pre-allocate 1024 huge pages of 2MB each (for a total of 2GB): "hugepagesz=2M hugepages=1024"

More detailed instructions can be found in the DPDK documentation (https://buildmedia.readthedocs.org/media/pdf/dpdk/latest/dpdk.pdf).

NFVbench Installation and Quick Start Guide

Make sure you satisfy the *hardware and software requirements <requirements>* before you start .

NFVbench can be used in CLI mode or in REST server mode. The CLI mode allows to run NFVbench benchmarks from the CLI. The REST server mode allows to run NFVbench benchmarks through a REST interface.

1. Container installation

To pull the latest NFVbench container image:

```
docker pull opnfv/nfvbench
```

2. NFVbench configuration file

Create a directory under $HOME called nfvbench to store the minimal configuration file:

```
mkdir $HOME/nfvbench
```

Create a new file containing the minimal configuration for NFVbench, we can call it any name, for example "nfvbench.cfg" and paste the following yaml template in the file:

```
openrc_file: /tmp/nfvbench/openrc
traffic_generator:
    generator_profile:
        - name: trex-local
          tool: TRex
          ip: 127.0.0.1
          cores: 3
          software_mode: false
          interfaces:
            - port: 0
              pci: "0a:00.0"
            - port: 1
              pci: "0a:00.1"
          intf_speed:
```

If OpenStack is not used, the openrc_file property can be removed.

If OpenStack is used, the openrc_file property must contain a valid container pathname of the OpenStack `openrc` file to connect to OpenStack using the OpenStack API. This file can be downloaded from the OpenStack Horizon dashboard (refer to the OpenStack documentation on how to retrieve the openrc file). This property must point to a valid pathname in the container (/tmp/nfvbench/openrc). We will map the host $HOME/nfvbench directory to the container /tmp/nfvbench directory and name the file "openrc". The file name viewed from the container will be "/tmp/nfvbench/openrc" (see container file pathname mapping in the next sections).

The PCI address of the 2 physical interfaces that will be used by the traffic generator must be configured. The PCI address can be obtained for example by using the "lspci" Linux command. For example:

```
[root@sjc04-pod6-build ~]# lspci | grep 710
0a:00.0 Ethernet controller: Intel Corporation Ethernet Controller X710 for 10GbE␣
↪SFP+ (rev 01)
0a:00.1 Ethernet controller: Intel Corporation Ethernet Controller X710 for 10GbE␣
↪SFP+ (rev 01)
0a:00.2 Ethernet controller: Intel Corporation Ethernet Controller X710 for 10GbE␣
↪SFP+ (rev 01)
0a:00.3 Ethernet controller: Intel Corporation Ethernet Controller X710 for 10GbE␣
↪SFP+ (rev 01)
```

In the above example, the PCI addresses "0a:00.0" and "0a:00.1" (first 2 ports of the quad port NIC) are used.

Warning: You have to put quotes around the pci addresses as shown in the above example, otherwise TRex will read it wrong. The other fields in the minimal configuration must be present and must have the same values as above.

3. Starting NFVbench in CLI mode

In this mode, the NFVbench code will reside in a container running in the background. This container will not run anything in the background. An alias is then used to invoke a new NFVbench benchmark run using docker exec. The $HOME/nfvbench directory on the host is mapped on the /tmp/nfvbench in the container to facilitate file sharing between the 2 environments.

Start NFVbench container

The NFVbench container can be started using docker run command or using docker compose.

To run NFVBench in CLI mode using docker run:

```
docker run --name nfvbench --detach --privileged -v /lib/modules/$(uname -r):/lib/
↪modules/$(uname -r) -v /usr/src/kernels:/usr/src/kernels -v /dev:/dev -v $HOME/
↪nfvbench:/tmp/nfvbench opnfv/nfvbench
```

Docker options	Description
–name nfvbench	container name is “nfvbench”
–detach	run container in background
–privileged	(optional) required if SELinux is enabled on the host
-v /lib/modules:/lib/modules	needed by kernel modules in the container
-v /usr/src/kernels:/usr/src/kernels	needed by TRex to build kernel modules when needed
-v /dev:/dev	needed by kernel modules in the container
-v $HOME/nfvbench:/tmp/nfvbench	folder mapping to pass files between the host and the docker space (see examples below) Here we map the $HOME/nfvbench directory on the host to the /tmp/nfvbench director in the container. Any other mapping can work as well
opnfv/nfvbench	container image name

To run NFVbench using docker compose, create the docker-compose.yml file and paste the following content:

```
version: '3'
services:
    nfvbench:
        image: "opnfv/nfvbench"
        container_name: "nfvbench"
        volumes:
            - /dev:/dev
            - /usr/src/kernels:/usr/src/kernels
            - /lib/modules:/lib/modules
            - ${HOME}/nfvbench:/tmp/nfvbench
        network_mode: "host"
        privileged: true
```

Then start the container in detached mode:

```
docker-compose up -d
```

Requesting an NFVbench benchmark run

Create an alias to make it easy to execute nfvbench commands directly from the host shell prompt:

```
alias nfvbench='docker exec -it nfvbench nfvbench'
```

The next to last "nfvbench" refers to the name of the container while the last "nfvbench" refers to the NFVbench binary that is available to run inside the container.

Once the alias is set, NFVbench runs can simply be requested from teh command line using "nfvbench <options>".

To verify it is working:

```
nfvbench --version
nfvbench --help
```

Example of run

To do a single run at 10,000pps bi-directional (or 5kpps in each direction) using the PVP packet path:

```
nfvbench -c /tmp/nfvbench/nfvbench.cfg --rate 10kpps
```

NFVbench options used:

- `-c /tmp/nfvbench/nfvbench.cfg` : specify the config file to use
- `--rate 10kpps` : specify rate of packets for test for both directions using the kpps unit (thousands of packets per second)

Retrieve complete configuration file as template

The full configuration file template with comments (yaml format) can be obtained using the –show-default-config option in order to use more advanced configuration options:

```
nfvbench --show-default-config > $HOME/nfvbench/full_nfvbench.cfg
```

Edit the full_nfvbench.cfg file to only keep those properties that need to be modified (preserving the nesting).

4. Start NFVbench in REST server mode

In this mode, the NFVbench REST server will run in the container. The $HOME/nfvbench directory on the host is mapped on the /tmp/nfvbench in the container to facilitate file sharing between the 2 environments.

Start NFVbench container

To start the NFVbench container with REST server using docker run cli:

```
docker run --name nfvbench --detach --privileged --net=host -e CONFIG_FILE="/tmp/
↪nfvbench/nfvbench.cfg" -v /lib/modules/$(uname -r):/lib/modules/$(uname -r) -v /usr/
↪src/kernels:/usr/src/kernels -v /dev:/dev -v $HOME/nfvbench:/tmp/nfvbench opnfv/
↪nfvbench start_rest_server
```

REST mode requires the same arguments as CLI mode and adds the following options:
+——————————————————-+——————————————————-+ | Docker options | Description | +===+===+ | –net=host | use “host” docker networking mode | | | Other modes (such as NAT) could be used if required | | | with proper adjustment of the port to use for REST | +——————————————————-+——————————————————-+ | -e CONFIG_FILE=”/tmp/nfvbench/nfvbench.cfg” | (optional) | | | specify the initial NFVbench config file to use. | | | defaults to none | +——————————————————-+——————————————————-+ | start_rest_server | to request a REST server to run in background in the | | | container | +——————————————————-+——————————————————-+ | -e HOST=”127.0.0.1” | (optional) | | | specify the IP address to listen to. | | | defaults to 127.0.0.1 | +——————————————————-+——————————————————-+ | -e PORT=7555 | (optional) | | | specify the port address to listen to. | | | defaults to 7555 | +——————————————————-+——————————————————-+

The initial configuration file is optional but is handy to define mandatory deployment parameters that are common to all subsequent REST requests. If this initial configuration file is not passed at container start time, it must be included in every REST request.

To start the NFVbench container with REST server using docker compose, use the following compose file:

```
version: '3'
services:
    nfvbench:
        image: "opnfv/nfvbench"
        container_name: "nfvbench_server"
        command: start_rest_server
        volumes:
            - /dev:/dev
            - /usr/src/kernels:/usr/src/kernels
            - /lib/modules:/lib/modules
            - ${HOME}/nfvbench:/tmp/nfvbench
        network_mode: "host"
        environment:
            - HOST="127.0.0.1"
            - PORT=7555
        privileged: true
```

Requesting an NFVbench benchmark run

To request a benchmark run, you must create a JSON document that describes the benchmark and send it to the NFVbench server in the body of a POST request.

Examples of REST requests

In this example, we will use curl to interact with the NFVbench REST server.

Query the NFVbench version:

```
[root@sjc04-pod3-mgmt ~]# curl -G http://127.0.0.1:7555/version
3.1.1
```

This is the JSON for a fixed rate run at 10,000pps bi-directional (or 5kpps in each direction) using the PVP packet path:

```
{"rate": "10kpps"}
```

This is the curl request to send this benchmark request to the NFVbench server:

```
[root@sjc04-pod3-mgmt ~]# curl -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST -d '{"rate": "10kpps"}' http://127.0.0.1:7555/start_run
{
  "error_message": "nfvbench run still pending",
  "status": "PENDING"
}
[root@sjc04-pod3-mgmt ~]#
```

This request will return immediately with status set to "PENDING" if the request was started successfully.

The status can be polled until the run completes. Here the poll returns a "PENDING" status, indicating the run is still not completed:

```
[root@sjc04-pod3-mgmt ~]# curl -G http://127.0.0.1:7555/status
{
  "error_message": "nfvbench run still pending",
  "status": "PENDING"
}
[root@sjc04-pod3-mgmt ~]#
```

Finally, the status request returns a "OK" status along with the full results (truncated here):

```
[root@sjc04-pod3-mgmt ~]# curl -G http://127.0.0.1:7555/status
{
  "result": {
    "benchmarks": {
      "network": {
        "service_chain": {
          "PVP": {
            "result": {
              "bidirectional": true,

...

  "status": "OK"
}
[root@sjc04-pod3-mgmt ~]#
```

Retrieve complete configuration file as template

7. Terminating the NFVbench container

When no longer needed, the container can be terminated using the usual docker commands:

```
docker kill nfvbench
docker rm nfvbench
```

7.1.4 Example of Results

Example run for fixed rate

```
nfvbench -c /nfvbench/nfvbenchconfig.json --rate 1%
```

```
========== NFVBench Summary ==========
Date: 2017-09-21 23:57:44
NFVBench version 1.0.9
Openstack Neutron:
  vSwitch: BASIC
  Encapsulation: BASIC
Benchmarks:
> Networks:
  > Components:
    > TOR:
        Type: None
    > Traffic Generator:
        Profile: trex-local
        Tool: TRex
    > Versions:
      > TOR:
      > Traffic Generator:
          build_date: Aug 30 2017
          version: v2.29
          built_by: hhaim
          build_time: 16:43:55
  > Service chain:
    > PVP:
      > Traffic:
          Profile: traffic_profile_64B
          Bidirectional: True
          Flow count: 10000
          Service chains count: 1
          Compute nodes: []

            Run Summary:

              +-----------------+-------------+----------------------+----------------
↪------+----------------------+
              |   L2 Frame Size |  Drop Rate  |   Avg Latency (usec) |   Min Latency␣
↪(usec) |   Max Latency (usec) |
              ␣
↪+=================+=============+======================+======================+=================
              |              64 |   0.0000%   |                   53 |              ␣
↪   20 |                  211 |
              +-----------------+-------------+----------------------+----------------
↪------+----------------------+


            L2 frame size: 64
            Chain analysis duration: 60.076 seconds

            Run Config:

              +-------------+---------------------------+-----------------------+----
↪-------------+---------------------------+-----------------------+-----------------
↪+
```

(continues on next page)

(continued from previous page)

```
|  Direction  |  Requested TX Rate (bps)  |  Actual TX Rate (bps)  |  RX Rate (bps)  |  Requested TX Rate (pps)  |  Actual TX Rate (pps)  |  RX Rate (pps)  |
+=============+===========================+========================+=================+===========================+========================+=================+
|   Forward   |       100.0000 Mbps       |      95.4546 Mbps      |  95.4546 Mbps   |        148,809 pps        |      142,045 pps       |   142,045 pps   |
+-------------+---------------------------+------------------------+-----------------+---------------------------+------------------------+-----------------+
|   Reverse   |       100.0000 Mbps       |      95.4546 Mbps      |  95.4546 Mbps   |        148,809 pps        |      142,045 pps       |   142,045 pps   |
+-------------+---------------------------+------------------------+-----------------+---------------------------+------------------------+-----------------+
|    Total    |       200.0000 Mbps       |     190.9091 Mbps      | 190.9091 Mbps   |        297,618 pps        |      284,090 pps       |   284,090 pps   |
+-------------+---------------------------+------------------------+-----------------+---------------------------+------------------------+-----------------+

            Chain Analysis:

+-------------------+----------+-----------------+---------------+---------------+---------+-----------------+---------------+---------------+
|     Interface     |  Device  |  Packets (fwd)  |  Drops (fwd)  |  Drop% (fwd)  |  Packets (rev)  |  Drops (rev)  |  Drop% (rev)  |
+===================+==========+=================+===============+===============+=================+===============+===============+
| traffic-generator |   trex   |    8,522,729    |               |               |    8,522,729    |             0 |    0.0000%    |
+-------------------+----------+-----------------+---------------+---------------+-----------------+---------------+---------------+
| traffic-generator |   trex   |    8,522,729    |               |               |    8,522,729    |             0 |    0.0000%    |
+-------------------+----------+-----------------+---------------+---------------+-----------------+---------------+---------------+
```

Example run for NDR/PDR with package size 1518B

```
nfvbench -c /nfvbench/nfvbenchconfig.json -fs 1518
```

```
========== NFVBench Summary ==========
Date: 2017-09-22 00:02:07
NFVBench version 1.0.9
Openstack Neutron:
  vSwitch: BASIC
  Encapsulation: BASIC
Benchmarks:
> Networks:
  > Components:
    > TOR:
        Type: None
```

(continues on next page)

(continued from previous page)

```
    > Traffic Generator:
        Profile: trex-local
        Tool: TRex
    > Versions:
      > TOR:
      > Traffic Generator:
          build_date: Aug 30 2017
          version: v2.29
          built_by: hhaim
          build_time: 16:43:55
  > Measurement Parameters:
      NDR: 0.001
      PDR: 0.1
  > Service chain:
    > PVP:
      > Traffic:
          Profile: custom_traffic_profile
          Bidirectional: True
          Flow count: 10000
          Service chains count: 1
          Compute nodes: []

            Run Summary:

              +-----+-----------------+------------------+------------------+---------
↪--------+----------------------+----------------------+----------------------+
              |  -  |   L2 Frame Size |  Rate (fwd+rev)  |  Rate (fwd+rev)  |  Avg␣
↪Drop Rate  |   Avg Latency (usec) |   Min Latency (usec) |  Max Latency (usec)  |
            ␣
↪+=====+=================+==================+==================+=================+=================
              | NDR |            1518 |   19.9805 Gbps   |  1,623,900 pps   |     0.
↪0001%     |                  342 |                   30 |         704          |
              +-----+-----------------+------------------+------------------+---------
↪--------+----------------------+----------------------+----------------------+
              | PDR |            1518 |   20.0000 Gbps   |  1,625,486 pps   |     0.
↪0022%     |                  469 |                   40 |        1,266         |
              +-----+-----------------+------------------+------------------+---------
↪--------+----------------------+----------------------+----------------------+


            L2 frame size: 1518
            Chain analysis duration: 660.442 seconds
            NDR search duration: 660 seconds
            PDR search duration: 0 seconds
```

7.1.5 Advanced Usage

This section covers a few examples on how to run NFVbench with multiple different settings. Below are shown the most common and useful use-cases and explained some fields from a default config file.

How to change any NFVbench run configuration (CLI)

NFVbench always starts with a default configuration which can further be refined (overridden) by the user from the CLI or from REST requests.

At first have a look at the default config:

```
nfvbench --show-default-config
```

It is sometimes useful derive your own configuration from a copy of the default config:

```
nfvbench --show-default-config > nfvbench.cfg
```

At this point you can edit the copy by:

- removing any parameter that is not to be changed (since NFVbench will always load the default configuration, default values are not needed)
- edit the parameters that are to be changed changed

A run with the new confguration can then simply be requested using the -c option and by using the actual path of the configuration file as seen from inside the container (in this example, we assume the current directory is mapped to /tmp/nfvbench in the container):

```
nfvbench -c /tmp/nfvbench/nfvbench.cfg
```

The same -c option also accepts any valid yaml or json string to override certain parameters without having to create a configuration file.

NFVbench provides many configuration options as optional arguments. For example the number of flows can be specified using the –flow-count option.

The flow count option can be specified in any of 3 ways:

- by providing a confguration file that has the flow_count value to use (-c myconfig.yaml and myconfig.yaml contains ‘flow_count: 100k’)
- by passing that yaml paremeter inline (-c “flow_count: 100k”) or (-c “{flow_count: 100k}”)
- by using the flow count optional argument (–flow-count 100k)

Showing the running configuration

Because configuration parameters can be overriden, it is sometimes useful to show the final configuration (after all oevrrides are done) by using the –show-config option. This final configuration is also called the “running” configuration.

For example, this will only display the running configuration (without actually running anything):

```
nfvbench -c "{flow_count: 100k, debug: true}" --show-config
```

Connectivity and Configuration Check

NFVbench allows to test connectivity to devices used with the selected packet path. It runs the whole test, but without actually sending any traffic. It is also a good way to check if everything is configured properly in the configuration file and what versions of components are used.

To verify everything works without sending any traffic, use the –no-traffic option:

```
nfvbench --no-traffic
```

Used parameters:

- `--no-traffic` or `-0` : sending traffic from traffic generator is skipped

TRex force restart

NFVbench allows to restart TRex traffic generator between runs. It runs the whole test, but restart TRex instance before generating new traffic.

To force restart, use the –restart option:

```
nfvbench --restart
```

Used parameters:

- `--restart` : restart traffic generator (TRex)

Rate Units

Parameter `--rate` accepts different types of values:

- packets per second (pps, kpps, mpps), e.g. `1000pps` or `10kpps`
- load percentage (%), e.g. `50%`
- bits per second (bps, kbps, Mbps, Gbps), e.g. `1Gbps`, `1000bps`
- NDR/PDR (ndr, pdr, ndr_pdr), e.g. `ndr_pdr`

NDR/PDR is the default rate when not specified.

Fixed Rate Run

Fixed rate run is the most basic type of NFVbench usage. It can be used to measure the drop rate with a fixed transmission rate of packets.

This example shows how to run the PVP packet path (which is the default packet path) with multiple different settings:

```
nfvbench -c nfvbench.cfg --no-cleanup --rate 100000pps --duration 30 --interval 15 --
↪json results.json
```

Used parameters:

- `-c nfvbench.cfg` : path to the config file
- `--no-cleanup` : resources (networks, VMs, attached ports) are not deleted after test is finished
- `--rate 100000pps` : defines rate of packets sent by traffic generator

- `--duration 30` : specifies how long should traffic be running in seconds
- `--interval 15` : stats are checked and shown periodically (in seconds) in this interval when traffic is flowing
- `--json results.json` : collected data are stored in this file after run is finished

Note: It is your responsibility to clean up resources if needed when `--no-cleanup` parameter is used. You can use the nfvbench_cleanup helper script for that purpose.

The `--json` parameter makes it easy to store NFVbench results. The –show-summary (or -ss) option can be used to display the results in a json results file in a text tabular format:

```
nfvbench --show-summary results.json
```

This example shows how to specify a different packet path:

```
nfvbench -c nfvbench.cfg --rate 1Mbps --inter-node --service-chain PVVP
```

Used parameters:

- `-c nfvbench.cfg` : path to the config file
- `--rate 1Mbps` : defines rate of packets sent by traffic generator
- `--inter-node` : VMs are created on different compute nodes, works only with PVVP flow
- `--service-chain PVVP` or `-sc PVVP` : specifies the type of service chain (or packet path) to use

Note: When parameter `--inter-node` is not used or there aren't enough compute nodes, VMs are on the same compute node.

A fixed rate run can also be used to check the running drop rate while traffic is being generated. In that case the –interval option can be used to specify the reporting interval in seconds (minimum is 1 second). This can be useful for example to see how packet drop rate evolves over time. One common use case is to see the drop rate when there is a network degradation (e.g. one of the 2 links in a bond goes down). The console output will show at every reporting interval the number of packets transmitted, received and estimated drop rate for the last reporting interval. The smaller is the interval the more precise is the drop rate.

Example of output where the reporting interval is set to 1 (second):

```
2020-04-25 12:59:16,618 INFO TX:   1,878,719,266; RX:   1,666,641,890; (Est.)␣
↪Dropped:            2; Drop rate:   0.0000%
2020-04-25 12:59:17,625 INFO TX:   1,883,740,078; RX:   1,671,662,706; (Est.)␣
↪Dropped:           -4; Drop rate:  -0.0001%
2020-04-25 12:59:18,632 INFO TX:   1,888,764,404; RX:   1,676,686,993; (Est.)␣
↪Dropped:           39; Drop rate:   0.0008%
2020-04-25 12:59:19,639 INFO TX:   1,893,785,063; RX:   1,681,276,714; (Est.)␣
↪Dropped:      430,938; Drop rate:   8.5833%
2020-04-25 12:59:20,645 INFO TX:   1,898,805,769; RX:   1,683,782,636; (Est.)␣
↪Dropped:    2,514,784; Drop rate:  50.0883%
2020-04-25 12:59:21,652 INFO TX:   1,903,829,191; RX:   1,686,289,860; (Est.)␣
↪Dropped:    2,516,198; Drop rate:  50.0893%
2020-04-25 12:59:22,658 INFO TX:   1,908,850,478; RX:   1,691,283,008; (Est.)␣
↪Dropped:       28,139; Drop rate:   0.5604%
2020-04-25 12:59:23,665 INFO TX:   1,913,870,692; RX:   1,696,301,242; (Est.)␣
↪Dropped:        1,980; Drop rate:   0.0394%
2020-04-25 12:59:24,672 INFO TX:   1,918,889,696; RX:   1,698,806,224; (Est.)␣
↪Dropped:    2,514,022; Drop rate:  50.0901%
```

(continues on next page)

(continued from previous page)

```
2020-04-25 12:59:25,680 INFO TX:   1,923,915,470; RX:   1,701,314,663; (Est.)␣
↪Dropped:    2,517,335; Drop rate:  50.0885%
2020-04-25 12:59:26,687 INFO TX:   1,928,944,879; RX:   1,705,886,869; (Est.)␣
↪Dropped:      457,203; Drop rate:   9.0906%
2020-04-25 12:59:27,696 INFO TX:   1,933,969,377; RX:   1,710,911,346; (Est.)␣
↪Dropped:           21; Drop rate:   0.0004%
2020-04-25 12:59:28,702 INFO TX:   1,938,998,536; RX:   1,713,843,740; (Est.)␣
↪Dropped:    2,096,765; Drop rate:  41.6922%
2020-04-25 12:59:29,710 INFO TX:   1,944,019,920; RX:   1,718,226,356; (Est.)␣
↪Dropped:      638,768; Drop rate:  12.7210%
2020-04-25 12:59:30,718 INFO TX:   1,949,050,206; RX:   1,723,256,639; (Est.)␣
↪Dropped:            3; Drop rate:   0.0001%
2020-04-25 12:59:31,725 INFO TX:   1,954,075,270; RX:   1,728,281,726; (Est.)␣
↪Dropped:          -23; Drop rate:  -0.0005%
2020-04-25 12:59:32,732 INFO TX:   1,959,094,908; RX:   1,733,301,290; (Est.)␣
↪Dropped:           74; Drop rate:   0.0015%
2020-04-25 12:59:33,739 INFO TX:   1,964,118,902; RX:   1,738,325,357; (Est.)␣
↪Dropped:          -73; Drop rate:  -0.0015%
2020-04-25 12:59:34,746 INFO TX:   1,969,143,790; RX:   1,743,350,230; (Est.)␣
↪Dropped:           15; Drop rate:   0.0003%
2020-04-25 12:59:35,753 INFO TX:   1,974,165,773; RX:   1,748,372,291; (Est.)␣
↪Dropped:          -78; Drop rate:  -0.0016%
2020-04-25 12:59:36,759 INFO TX:   1,979,188,496; RX:   1,753,394,957; (Est.)␣
↪Dropped:           57; Drop rate:   0.0011%
2020-04-25 12:59:37,767 INFO TX:   1,984,208,956; RX:   1,757,183,844; (Est.)␣
↪Dropped:    1,231,573; Drop rate:  24.5311%
2020-04-25 12:59:38,773 INFO TX:   1,989,233,595; RX:   1,761,729,705; (Est.)␣
↪Dropped:      478,778; Drop rate:   9.5286%
2020-04-25 12:59:39,780 INFO TX:   1,994,253,350; RX:   1,766,749,467; (Est.)␣
↪Dropped:           -7; Drop rate:  -0.0001%
2020-04-25 12:59:40,787 INFO TX:   1,999,276,622; RX:   1,771,772,738; (Est.)␣
↪Dropped:            1; Drop rate:   0.0000%
2020-04-25 12:59:41,794 INFO TX:   2,004,299,940; RX:   1,776,796,065; (Est.)␣
↪Dropped:           -9; Drop rate:  -0.0002%
2020-04-25 12:59:42,800 INFO TX:   2,009,320,453; RX:   1,781,816,583; (Est.)␣
↪Dropped:           -5; Drop rate:  -0.0001%
2020-04-25 12:59:43,807 INFO TX:   2,014,340,581; RX:   1,786,503,172; (Est.)␣
↪Dropped:      333,539; Drop rate:   6.6440%
2020-04-25 12:59:44,814 INFO TX:   2,019,362,996; RX:   1,789,009,857; (Est.)␣
↪Dropped:    2,515,730; Drop rate:  50.0900%
2020-04-25 12:59:45,821 INFO TX:   2,024,386,346; RX:   1,791,517,070; (Est.)␣
↪Dropped:    2,516,137; Drop rate:  50.0888%
```

How to read each line:

```
2020-04-25 10:46:41,276 INFO TX:       4,004,436; RX:       4,004,381; (Est.)␣
↪Dropped:           55; Drop rate:   0.0014%
```

At this poing in time, NFvbench has sent 4,004,436 and received 4,004,381 since the start of the run. There is deficit of 55 packets on reception which corresponds to 0.0014% of all packets sent during that reporting window interval (last 1 second) A negative value means that the RX count is higher than the tx count in that window – this is possible since the RX and TX reads are not atomic.

NDR and PDR

The NDR and PDR test is used to determine the maximum throughput performance of the system under test following guidelines defined in RFC-2544:

- NDR (No Drop Rate): maximum packet rate sent without dropping any packet
- PDR (Partial Drop Rate): maximum packet rate sent while allowing a given maximum drop rate

The NDR search can also be relaxed to allow some very small amount of drop rate (lower than the PDR maximum drop rate). NFVbench will measure the NDR and PDR values by driving the traffic generator through multiple iterations at different transmission rates using a binary search algorithm.

The configuration file contains section where settings for NDR/PDR can be set.

```
# NDR/PDR configuration
measurement:
    # Drop rates represent the ratio of dropped packet to the total number of packets␣
↪sent.
    # Values provided here are percentages. A value of 0.01 means that at most 0.01%␣
↪of all
    # packets sent are dropped (or 1 packet every 10,000 packets sent)

    # No Drop Rate; Default to 0.001%
    NDR: 0.001
    # Partial Drop Rate; NDR should always be less than PDR
    PDR: 0.1
    # The accuracy of NDR and PDR load percentiles; The actual load percentile that␣
↪match NDR
    # or PDR should be within `load_epsilon` difference than the one calculated.
    load_epsilon: 0.1
```

Because NDR/PDR is the default `--rate` value, it is possible to run NFVbench simply like this:

```
nfvbench -c nfvbench.cfg
```

Other possible run options:

```
nfvbench -c nfvbench.cfg --duration 120 --json results.json
```

Used parameters:

- `-c nfvbench.cfg` : path to the config file
- `--duration 120` : specifies how long should be traffic running in each iteration
- `--json results.json` : collected data are stored in this file after run is finished

Multichain

NFVbench allows to run multiple chains at the same time. For example it is possible to stage the PVP service chain N-times, where N can be as much as your compute power can scale. With N = 10, NFVbench will spawn 10 VMs as a part of 10 simultaneous PVP chains.

The number of chains is specified by `--service-chain-count` or `-scc` flag with a default value of 1. For example to run NFVbench with 3 PVP chains:

```
nfvbench -c nfvbench.cfg --rate 10000pps -scc 3
```

It is not necessary to specify the service chain type (-sc) because PVP is set as default. The PVP service chains will have 3 VMs in 3 chains with this configuration. If `-sc PVVP` is specified instead, there would be 6 VMs in 3 chains as this service chain has 2 VMs per chain. Both **single run** or **NDR/PDR** can be run as multichain. Running multichain is a scenario closer to a real life situation than runs with a single chain.

Multiflow

NFVbench always generates L3 packets from the traffic generator but allows the user to specify how many flows to generate. A flow is identified by a unique src/dest MAC IP and port tuple that is sent by the traffic generator. Flows are generated by ranging the IP adresses but using a small fixed number of MAC addresses.

The number of flows will be spread roughly even between chains when more than 1 chain is being tested. For example, for 11 flows and 3 chains, number of flows that will run for each chain will be 3, 4, and 4 flows respectively.

The number of flows is specified by `--flow-count` or `-fc` flag, the default value is 2 (1 flow in each direction). To run NFVbench with 3 chains and 100 flows, use the following command:

```
nfvbench -c nfvbench.cfg --rate 10000pps -scc 3 -fc 100
```

Note that from a vswitch point of view, the number of flows seen will be higher as it will be at least 4 times the number of flows sent by the traffic generator (add flow to VM and flow from VM).

IP addresses generated can be controlled with the following NFVbench configuration options:

```
ip_addrs: ['10.0.0.0/8', '20.0.0.0/8']
ip_addrs_step: 0.0.0.1
tg_gateway_ip_addrs: ['1.1.0.100', '2.2.0.100']
tg_gateway_ip_addrs_step: 0.0.0.1
gateway_ip_addrs: ['1.1.0.2', '2.2.0.2']
gateway_ip_addrs_step: 0.0.0.1
```

`ip_addrs` are the start of the 2 ip address ranges used by the traffic generators as the packets source and destination packets where each range is associated to virtual devices simulated behind 1 physical interface of the traffic generator. These can also be written in CIDR notation to represent the subnet.

`tg_gateway_ip_addrs` are the traffic generator gateway (virtual) ip addresses, all traffic to/from the virtual devices go through them.

`gateway_ip_addrs` are the 2 gateway ip address ranges of the VMs used in the external chains. They are only used with external chains and must correspond to their public IP address.

The corresponding `step` is used for ranging the IP addresses from the *ip_addrs*`, `tg_gateway_ip_addrs` and `gateway_ip_addrs` base addresses. 0.0.0.1 is the default step for all IP ranges. In `ip_addrs`, 'random' can be configured which tells NFVBench to generate random src/dst IP pairs in the traffic stream.

UDP ports can be controlled with the following NFVbench configuration options:

```
udp_src_port: ['1024', '65000']
udp_dst_port: 53
udp_port_step: '1'
```

`udp_src_port` and `udp_dst_port` are the UDP port value used by the traffic generators. These can be written for unique port or range ports for all flow.

The corresponding `udp_port_step` is used for ranging the UDP port. 1 is the default step for all UDP ranges, 'random' can be configured which tells NFVBench to generate random src/dst UDP pairs in the traffic stream.

NB: Use of UDP range will increase possible values of flows (based on ip src/dst and port src/dst tuple). NFVBench will calculate the least common multiple for this tuple to adapt flows generation to `flow_count` parameter.

Examples of multiflow

1. Source IP is static and one UDP port used (default configuration)

NFVbench configuration options:

```
ip_addrs: ['110.0.0.0/8', '120.0.0.0/8']
ip_addrs_step: 0.0.0.1
tg_gateway_ip_addrs: ['1.1.0.100', '2.2.0.100']
tg_gateway_ip_addrs_step: 0.0.0.1
gateway_ip_addrs: ['1.1.0.2', '2.2.0.2']
gateway_ip_addrs_step: 0.0.0.1
```

To run NFVbench with 3 chains and 100 flows, use the following command:

```
nfvbench -c nfvbench.cfg --rate 10000pps -scc 3 -fc 100
```

The least common multiple for this configuration is lcm(16 777 216, 16 777 216, 1, 1) = 16 777 216. .. note:: LCM method used IP pools sizes and UDP source and destination range sizes

Requested flow count is lower than configuration capacity. So, NFVbench will limit IP range to generate accurate flows:

```
2020-06-17 07:37:47,012 INFO Port 0, chain 0: IP src range [110.0.0.0,110.0.0.0]
2020-06-17 07:37:47,012 INFO Port 0, chain 0: IP dst range [120.0.0.0,120.0.0.15]
2020-06-17 07:37:47,012 INFO Port 0, chain 0: UDP src range [53,53]
2020-06-17 07:37:47,012 INFO Port 0, chain 0: UDP dst range [53,53]
2020-06-17 07:37:47,015 INFO Port 1, chain 0: IP src range [120.0.0.0,120.0.0.0]
2020-06-17 07:37:47,015 INFO Port 1, chain 0: IP dst range [110.0.0.0,110.0.0.15]
2020-06-17 07:37:47,015 INFO Port 1, chain 0: UDP src range [53,53]
2020-06-17 07:37:47,015 INFO Port 1, chain 0: UDP dst range [53,53]

2020-06-17 07:38:47,012 INFO Port 0, chain 1: IP src range [110.0.0.1,110.0.0.1]
2020-06-17 07:38:47,012 INFO Port 0, chain 1: IP dst range [120.0.0.16,120.0.0.32]
2020-06-17 07:38:47,012 INFO Port 0, chain 1: UDP src range [53,53]
2020-06-17 07:38:47,012 INFO Port 0, chain 1: UDP dst range [53,53]
2020-06-17 07:38:47,015 INFO Port 1, chain 1: IP src range [120.0.0.1,120.0.0.1]
2020-06-17 07:38:47,015 INFO Port 1, chain 1: IP dst range [110.0.0.16,110.0.0.32]
2020-06-17 07:38:47,015 INFO Port 1, chain 1: UDP src range [53,53]
2020-06-17 07:38:47,015 INFO Port 1, chain 1: UDP dst range [53,53]

2020-06-17 07:39:47,012 INFO Port 0, chain 2: IP src range [110.0.0.2,110.0.0.2]
2020-06-17 07:39:47,012 INFO Port 0, chain 2: IP dst range [120.0.0.33,120.0.0.49]
2020-06-17 07:39:47,012 INFO Port 0, chain 2: UDP src range [53,53]
2020-06-17 07:39:47,012 INFO Port 0, chain 2: UDP dst range [53,53]
2020-06-17 07:39:47,015 INFO Port 1, chain 2: IP src range [120.0.0.2,120.0.0.2]
2020-06-17 07:39:47,015 INFO Port 1, chain 2: IP dst range [110.0.0.33,110.0.0.49]
2020-06-17 07:39:47,015 INFO Port 1, chain 2: UDP src range [53,53]
2020-06-17 07:39:47,015 INFO Port 1, chain 2: UDP dst range [53,53]
```

2. Source IP is static, IP step is random and one UDP port used

NFVbench configuration options:

```
ip_addrs: ['110.0.0.0/8', '120.0.0.0/8']
ip_addrs_step: 'random'
tg_gateway_ip_addrs: ['1.1.0.100', '2.2.0.100']
tg_gateway_ip_addrs_step: 0.0.0.1
```

(continues on next page)

(continued from previous page)

```
gateway_ip_addrs: ['1.1.0.2', '2.2.0.2']
gateway_ip_addrs_step: 0.0.0.1
```

To run NFVbench with 3 chains and 100 flows, use the following command:

```
nfvbench -c nfvbench.cfg --rate 10000pps -scc 3 -fc 100
```

The least common multiple for this configuration is lcm(16 777 216, 16 777 216, 1, 1) = 16 777 216. .. note:: LCM method used IP pools sizes and UDP source and destination range sizes

Requested flow count is lower than configuration capacity. So, NFVbench will limit IP range to generate accurate flows:

```
2020-06-17 07:37:47,012 INFO Port 0, chain 0: IP src range [110.0.0.0,110.0.0.0]
2020-06-17 07:37:47,012 INFO Port 0, chain 0: IP dst range [120.0.0.0,120.0.0.15]
2020-06-17 07:37:47,012 INFO Port 0, chain 0: UDP src range [53,53]
2020-06-17 07:37:47,012 INFO Port 0, chain 0: UDP dst range [53,53]
2020-06-17 07:37:47,015 INFO Port 1, chain 0: IP src range [120.0.0.0,120.0.0.0]
2020-06-17 07:37:47,015 INFO Port 1, chain 0: IP dst range [110.0.0.0,110.0.0.15]
2020-06-17 07:37:47,015 INFO Port 1, chain 0: UDP src range [53,53]
2020-06-17 07:37:47,015 INFO Port 1, chain 0: UDP dst range [53,53]

2020-06-17 07:38:47,012 INFO Port 0, chain 1: IP src range [110.0.0.1,110.0.0.1]
2020-06-17 07:38:47,012 INFO Port 0, chain 1: IP dst range [120.0.0.16,120.0.0.32]
2020-06-17 07:38:47,012 INFO Port 0, chain 1: UDP src range [53,53]
2020-06-17 07:38:47,012 INFO Port 0, chain 1: UDP dst range [53,53]
2020-06-17 07:38:47,015 INFO Port 1, chain 1: IP src range [120.0.0.1,120.0.0.1]
2020-06-17 07:38:47,015 INFO Port 1, chain 1: IP dst range [110.0.0.16,110.0.0.32]
2020-06-17 07:38:47,015 INFO Port 1, chain 1: UDP src range [53,53]
2020-06-17 07:38:47,015 INFO Port 1, chain 1: UDP dst range [53,53]

2020-06-17 07:39:47,012 INFO Port 0, chain 2: IP src range [110.0.0.2,110.0.0.2]
2020-06-17 07:39:47,012 INFO Port 0, chain 2: IP dst range [120.0.0.33,120.0.0.49]
2020-06-17 07:39:47,012 INFO Port 0, chain 2: UDP src range [53,53]
2020-06-17 07:39:47,012 INFO Port 0, chain 2: UDP dst range [53,53]
2020-06-17 07:39:47,015 INFO Port 1, chain 2: IP src range [120.0.0.2,120.0.0.2]
2020-06-17 07:39:47,015 INFO Port 1, chain 2: IP dst range [110.0.0.33,110.0.0.49]
2020-06-17 07:39:47,015 INFO Port 1, chain 2: UDP src range [53,53]
2020-06-17 07:39:47,015 INFO Port 1, chain 2: UDP dst range [53,53]
2020-06-17 07:39:47,015 WARNING Using random step, the number of flows can be less␣
↪than the requested number of flows due to repeatable multivariate random generation␣
↪which can reproduce the same pattern of values
```

By using a random step the number of generated flows may be less than the number of requested flows. This is due to the probability of drawing the same value several times (Bernouillian drawing) from the IP range used and thus generating the same flow sequence. By using a high range of UDP ports couple with `udp_port_step='random'` the probability to reach the requested flow counts is greater. As latency stream is a separate stream than data one and have his own random draw, NFVbench will use only one packet signature (same IP and ports used for all latency packets) to avoid flow count overflow. So in some cases, generated flow count can be equal to the requested flow count + 1 (latency stream).

For deterministic flow count we recommend to use a step different from random.

3. Source IP is static, IP step is 5 and one UDP port used

NFVbench configuration options:

```
ip_addrs: ['110.0.0.0/8', '120.0.0.0/8']
ip_addrs_step: '0.0.0.5'
tg_gateway_ip_addrs: ['1.1.0.100', '2.2.0.100']
tg_gateway_ip_addrs_step: 0.0.0.1
gateway_ip_addrs: ['1.1.0.2', '2.2.0.2']
gateway_ip_addrs_step: 0.0.0.1
```

To run NFVbench with 3 chains and 100 flows, use the following command:

```
nfvbench -c nfvbench.cfg --rate 10000pps -scc 3 -fc 100
```

The least common multiple for this configuration is lcm(16 777 216, 16 777 216, 1, 1) = 16 777 216. .. note:: LCM method used IP pools sizes and UDP source and destination range sizes

Requested flow count is lower than configuration capacity. So, NFVbench will limit IP range to generate accurate flows:

```
2020-06-17 07:37:47,012 INFO Port 0, chain 0: IP src range [110.0.0.0,110.0.0.0]
2020-06-17 07:37:47,012 INFO Port 0, chain 0: IP dst range [120.0.0.0,120.0.0.75]
2020-06-17 07:37:47,012 INFO Port 0, chain 0: UDP src range [53,53]
2020-06-17 07:37:47,012 INFO Port 0, chain 0: UDP dst range [53,53]
2020-06-17 07:37:47,015 INFO Port 1, chain 0: IP src range [120.0.0.0,120.0.0.0]
2020-06-17 07:37:47,015 INFO Port 1, chain 0: IP dst range [110.0.0.0,110.0.0.75]
2020-06-17 07:37:47,015 INFO Port 1, chain 0: UDP src range [53,53]
2020-06-17 07:37:47,015 INFO Port 1, chain 0: UDP dst range [53,53]

2020-06-17 07:38:47,012 INFO Port 0, chain 1: IP src range [110.0.0.5,110.0.0.5]
2020-06-17 07:38:47,012 INFO Port 0, chain 1: IP dst range [120.0.0.80,120.0.0.160]
2020-06-17 07:38:47,012 INFO Port 0, chain 1: UDP src range [53,53]
2020-06-17 07:38:47,012 INFO Port 0, chain 1: UDP dst range [53,53]
2020-06-17 07:38:47,015 INFO Port 1, chain 1: IP src range [120.0.0.5,120.0.0.5]
2020-06-17 07:38:47,015 INFO Port 1, chain 1: IP dst range [110.0.0.80,110.0.0.160]
2020-06-17 07:38:47,015 INFO Port 1, chain 1: UDP src range [53,53]
2020-06-17 07:38:47,015 INFO Port 1, chain 1: UDP dst range [53,53]

2020-06-17 07:39:47,012 INFO Port 0, chain 2: IP src range [110.0.0.10,110.0.0.10]
2020-06-17 07:39:47,012 INFO Port 0, chain 2: IP dst range [120.0.0.165,120.0.0.245]
2020-06-17 07:39:47,012 INFO Port 0, chain 2: UDP src range [53,53]
2020-06-17 07:39:47,012 INFO Port 0, chain 2: UDP dst range [53,53]
2020-06-17 07:39:47,015 INFO Port 1, chain 2: IP src range [120.0.0.10,120.0.0.10]
2020-06-17 07:39:47,015 INFO Port 1, chain 2: IP dst range [110.0.0.165,110.0.0.245]
2020-06-17 07:39:47,015 INFO Port 1, chain 2: UDP src range [53,53]
2020-06-17 07:39:47,015 INFO Port 1, chain 2: UDP dst range [53,53]
```

4. Source IP is static, IP and UDP ranges sizes greater than requested flow count, UDP step is random

NFVbench configuration options:

```
ip_addrs: ['110.0.0.0/29', '120.0.0.0/30']
tg_gateway_ip_addrs: ['1.1.0.100', '2.2.0.100']
tg_gateway_ip_addrs_step: 0.0.0.1
gateway_ip_addrs: ['1.1.0.2', '2.2.0.2']
gateway_ip_addrs_step: 0.0.0.1
udp_src_port: ['10', '14']
udp_dst_port: ['20', '25']
udp_port_step: 'random'
```

To run NFVbench with 3 chains and 100 flows, use the following command:

```
nfvbench -c nfvbench.cfg --rate 10000pps -scc 3 -fc 100
```

The least common multiple for this configuration is lcm(8, 4, 5, 6) = 120. .. note:: LCM method used IP pools sizes and UDP source and destination range sizes

Requested flow count is higher than IP range (8 and 4 IP addresses available) and UDP (5 and 6 ports available) configuration capacity. As the combination of ranges does not permit to obtain an accurate flow count, NFVbench will override the *udp_port_step* property to '1' (was 'random') to allow flows creation. A warning log will appear to inform NFVbench user that step properties will be overriden So, NFVbench will determine each pool size to generate accurate flows:

```
2020-06-17 07:37:47,010 WARNING Current values of ip_addrs_step and/or udp_port_step␣
↪properties do not allow to control an accurate flow count. Values will be␣
↪overridden as follows:
2020-06-17 07:37:47,011 INFO udp_port_step='1' (previous value: udp_port_step='random
↪')
2020-06-17 07:37:47,012 INFO Port 0, chain 0: IP src range [110.0.0.0,110.0.0.0]
2020-06-17 07:37:47,012 INFO Port 0, chain 0: IP dst range [120.0.0.0,120.0.0.0]
2020-06-17 07:37:47,012 INFO Port 0, chain 0: UDP src range [10,14]
2020-06-17 07:37:47,012 INFO Port 0, chain 0: UDP dst range [20,25]
2020-06-17 07:37:47,013 WARNING Current values of ip_addrs_step and/or udp_port_step␣
↪properties do not allow to control an accurate flow count. Values will be␣
↪overridden as follows:
2020-06-17 07:37:47,013 INFO udp_port_step='1' (previous value: udp_port_step='random'
2020-06-17 07:37:47,015 INFO Port 1, chain 0: IP src range [120.0.0.0,120.0.0.0]
2020-06-17 07:37:47,015 INFO Port 1, chain 0: IP dst range [110.0.0.0,110.0.0.0]
2020-06-17 07:37:47,015 INFO Port 1, chain 0: UDP src range [10,14]
2020-06-17 07:37:47,015 INFO Port 1, chain 0: UDP dst range [20,25]

2020-06-17 07:38:47,010 WARNING Current values of ip_addrs_step and/or udp_port_step␣
↪properties do not allow to control an accurate flow count. Values will be␣
↪overridden as follows:
2020-06-17 07:38:47,011 INFO udp_port_step='1' (previous value: udp_port_step='random'
2020-06-17 07:38:47,012 INFO Port 0, chain 1: IP src range [110.0.0.1,110.0.0.1]
2020-06-17 07:38:47,012 INFO Port 0, chain 1: IP dst range [120.0.0.1,120.0.0.1]
2020-06-17 07:38:47,012 INFO Port 0, chain 1: UDP src range [10,14]
2020-06-17 07:38:47,012 INFO Port 0, chain 1: UDP dst range [20,25]
2020-06-17 07:38:47,013 WARNING Current values of ip_addrs_step and/or udp_port_step␣
↪properties do not allow to control an accurate flow count. Values will be␣
↪overridden as follows:
2020-06-17 07:38:47,013 INFO udp_port_step='1' (previous value: udp_port_step='random'
2020-06-17 07:38:47,015 INFO Port 1, chain 1: IP src range [120.0.0.1,120.0.0.1]
2020-06-17 07:38:47,015 INFO Port 1, chain 1: IP dst range [110.0.0.1,110.0.0.1]
2020-06-17 07:38:47,015 INFO Port 1, chain 1: UDP src range [10,14]
2020-06-17 07:38:47,015 INFO Port 1, chain 1: UDP dst range [20,25]

2020-06-17 07:39:47,010 WARNING Current values of ip_addrs_step and/or udp_port_step␣
↪properties do not allow to control an accurate flow count. Values will be␣
↪overridden as follows:
2020-06-17 07:39:47,011 INFO udp_port_step='1' (previous value: udp_port_step='random'
2020-06-17 07:39:47,012 INFO Port 0, chain 2: IP src range [110.0.0.2,110.0.0.2]
2020-06-17 07:39:47,012 INFO Port 0, chain 2: IP dst range [120.0.0.2,120.0.0.2]
2020-06-17 07:39:47,012 INFO Port 0, chain 2: UDP src range [10,14]
2020-06-17 07:39:47,012 INFO Port 0, chain 2: UDP dst range [20,25]
2020-06-17 07:39:47,013 WARNING Current values of ip_addrs_step and/or udp_port_step␣
↪properties do not allow to control an accurate flow count. Values will be␣
↪overridden as follows:
```

(continues on next page)

(continued from previous page)

```
2020-06-17 07:39:47,013 INFO udp_port_step='1' (previous value: udp_port_step='random'
2020-06-17 07:39:47,015 INFO Port 1, chain 2: IP src range [120.0.0.2,120.0.0.2]
2020-06-17 07:39:47,015 INFO Port 1, chain 2: IP dst range [110.0.0.2,110.0.0.2]
2020-06-17 07:39:47,015 INFO Port 1, chain 2: UDP src range [10,14]
2020-06-17 07:39:47,015 INFO Port 1, chain 2: UDP dst range [20,25]
```

Traffic Configuration via CLI

While traffic configuration can be modified using the configuration file, it can be inconvenient to have to change the configuration file everytime you need to change a traffic configuration option. Traffic configuration options can be overridden with a few CLI options.

Here is an example of configuring traffic via CLI:

```
nfvbench --rate 10kpps --service-chain-count 2 -fs 64 -fs IMIX -fs 1518 --unidir
```

This command will run NFVbench with a unidirectional flow for three packet sizes 64B, IMIX, and 1518B.

Used parameters:

- `--rate 10kpps` : defines rate of packets sent by traffic generator (total TX rate)
- `-scc 2` or `--service-chain-count 2` : specifies number of parallel chains of given flow to run (default to 1)
- `-fs 64` or `--frame-size 64`: add the specified frame size to the list of frame sizes to run
- `--unidir` : run traffic with unidirectional flow (default to bidirectional flow)

L2 loopback test via CLI

The CLI allows running a pure L2 loopback benchmark with the `--l2-loopback` option. Enabling this mode overrides any service chain type selected in the config file. The usual argument would be a single VLAN ID but the syntax has been extended.

Examples of runs:

- specify the vlan ID

```
nfvbench -c nfvbench.cfg --frame-size=64 --rate=100% --duration=10 --l2-
↪loopback=123
```

- specify a list of vlan IDs

  Several service chains are created, the count is adjusted to the list size.

```
nfvbench -c nfvbench.cfg -fs=64 --rate=100% --duration=10 --l2-loopback=123,124,
↪125
```

- enable the mode without VLAN tagging

  In this case the service chain count is forced to 1.

```
nfvbench -c nfvbench.cfg -fs=64 --rate=100% --duration=10 --l2-loopback=no-tag
```

- use different VLAN tags for left & right side ports

```
nfvbench -c nfvbench.cfg -fs=64 --rate=100% --duration=10 --l2-loopback=111_211
nfvbench -c nfvbench.cfg -fs=64 --rate=100% --duration=10 --l2-loopback=111,112_
↪211,212
```

Note: It may look bizarre to specify mismatched VLAN tags for left & right sides, however no assumption is made about the loop implementation. This could help testing some exotic L2 layer configuration comprising a VLAN rewriting.

- enable the mode, starting from current settings (prepared in the cfg file)

  In this case the service chain count is not adjusted.

```
nfvbench -c nfvbench.cfg -fs=64 --rate=100% --duration=10 --l2-loopback=true
```

- disable the mode (possibly enabled in the cfg file)

```
nfvbench -c nfvbench.cfg --l2-loopback=false
```

MAC Addresses

NFVbench will dicover the MAC addresses to use for generated frames using: - either OpenStack discovery (find the MAC of an existing VM) in the case of PVP and PVVP service chains - or using dynamic ARP discovery (find MAC from IP) in the case of external chains. - In case of L3 chain with SDN-GW or router between traffic generator and loop VM ARP is needed to discover SDN-GW mac addresses, use `--loop-vm-arp` flag or `loop_vm_arp: true` in config file.

Status and Cleanup of NFVbench Resources

The –status option will display the status of NFVbench and list any NFVbench resources. You need to pass the OpenStack RC file in order to connect to OpenStack.

```
# nfvbench --status -r /tmp/nfvbench/openrc
2018-04-09 17:05:48,682 INFO Version: 1.3.2.dev1
2018-04-09 17:05:48,683 INFO Status: idle
2018-04-09 17:05:48,757 INFO Discovering instances nfvbench-loop-vm...
2018-04-09 17:05:49,252 INFO Discovering flavor nfvbench.medium...
2018-04-09 17:05:49,281 INFO Discovering networks...
2018-04-09 17:05:49,365 INFO No matching NFVbench resources found
#
```

The Status can be either “idle” or “busy (run pending)”.

The –cleanup option will first discover resources created by NFVbench and prompt if you want to proceed with cleaning them up. Example of run:

```
# nfvbench --cleanup -r /tmp/nfvbench/openrc
2018-04-09 16:58:00,204 INFO Version: 1.3.2.dev1
2018-04-09 16:58:00,205 INFO Status: idle
2018-04-09 16:58:00,279 INFO Discovering instances nfvbench-loop-vm...
2018-04-09 16:58:00,829 INFO Discovering flavor nfvbench.medium...
2018-04-09 16:58:00,876 INFO Discovering networks...
2018-04-09 16:58:00,960 INFO Discovering ports...
2018-04-09 16:58:01,012 INFO Discovered 6 NFVbench resources:
```

(continues on next page)

(continued from previous page)

```
+----------+------------------+--------------------------------------+
| Type     | Name             | UUID                                 |
|----------+------------------+--------------------------------------|
| Instance | nfvbench-loop-vm0 | b039b858-777e-467e-99fb-362f856f4a94 |
| Flavor   | nfvbench.medium   | a027003c-ad86-4f24-b676-2b05bb06adc0 |
| Network  | nfvbench-net0     | bca8d183-538e-4965-880e-fd92d48bfe0d |
| Network  | nfvbench-net1     | c582a201-8279-4309-8084-7edd6511092c |
| Port     |                   | 67740862-80ac-4371-b04e-58a0b0f05085 |
| Port     |                   | b5db95b9-e419-4725-951a-9a8f7841e66a |
+----------+------------------+--------------------------------------+
2018-04-09 16:58:01,013 INFO NFVbench will delete all resources shown...
Are you sure? (y/n) y
2018-04-09 16:58:01,865 INFO Deleting instance nfvbench-loop-vm0...
2018-04-09 16:58:02,058 INFO     Waiting for 1 instances to be fully deleted...
2018-04-09 16:58:02,182 INFO     1 yet to be deleted by Nova, retries left=6...
2018-04-09 16:58:04,506 INFO     1 yet to be deleted by Nova, retries left=5...
2018-04-09 16:58:06,636 INFO     1 yet to be deleted by Nova, retries left=4...
2018-04-09 16:58:08,701 INFO Deleting flavor nfvbench.medium...
2018-04-09 16:58:08,729 INFO Deleting port 67740862-80ac-4371-b04e-58a0b0f05085...
2018-04-09 16:58:09,102 INFO Deleting port b5db95b9-e419-4725-951a-9a8f7841e66a...
2018-04-09 16:58:09,620 INFO Deleting network nfvbench-net0...
2018-04-09 16:58:10,357 INFO Deleting network nfvbench-net1...
#
```

The –force-cleanup option will do the same but without prompting for confirmation.

Service mode for TRex

The `--service-mode` option allows you to capture traffic on a TRex window during the NFVBench test. Thus, you will be able to capture packets generated by TRex to observe many information on it.

Example of use :

```
nfvbench ``--service-mode``
```

Note: It is preferable to define the minimum rate (2002 pps) to have a better capture

In another bash window, you should connect to the TRex console doing:

```
cd /opt/trex/vX.XX/ # use completion here to find your corresponding TRex version
./trex-console --python3 -r
# capture on port number 0
capture monitor start --rx 0 -v

# to stop capture
capture monitor stop
```

Start this capture once you have started the NFVBench test, and you will observe packets on the TRex console:

```
#26342 Port: 0 —— RX

trex(read-only)>

    Type: UDP, Size: 66 B, TS: 26.30 [sec]
```

(continues on next page)

(continued from previous page)

```
trex(read-only)>
    ###[ Ethernet ]###
        dst       = a0:36:9f:7a:58:8e
        src       = fa:16:3e:57:8f:df
        type      = 0x8100
    ###[ 802.1Q ]###
        prio      = 0
        id        = 0
        vlan      = 1093
        type      = 0x800
    ###[ IP ]###
        version   = 4
        ihl       = 5
        tos       = 0x1
        len       = 46
        id        = 65535
        flags     =
        frag      = 0
        ttl       = 63
        proto     = udp
        chksum    = 0x8425
        src       = 120.0.0.0
        dst       = 110.0.17.153
        \options   \
    ###[ UDP ]###
        sport     = 53
        dport     = 53
        len       = 26
        chksum    = 0xfd83
    ###[ Raw ]###
        load      = "xx\xab'\x01\x00?s\x00\x00\xbci\xf0_{U~\x00"
    ###[ Padding ]###
        load      = '6\x85'
```

Check on the NFVBench window that the following log appears just before the testing phase:

```
2019-10-21 09:38:51,532 INFO Starting to generate traffic...
2019-10-21 09:38:51,532 INFO Running traffic generator
2019-10-21 09:38:51,541 INFO ``Service mode is enabled``
2019-10-21 09:38:52,552 INFO TX: 2004; RX: 2003; Est. Dropped: 1; Est. Drop rate: 0.
↪0499%
2019-10-21 09:38:53,559 INFO TX: 4013; RX: 4011; Est. Dropped: 2; Est. Drop rate: 0.
↪0498%
```

Recording packet using service mode for TRex

Check on the NFVBench window that the following log appears just before the testing phase:

```
2019-10-21 09:38:51,532 INFO Starting to generate traffic...
2019-10-21 09:38:51,532 INFO Running traffic generator
2019-10-21 09:38:51,541 INFO ``Service mode is enabled``
2019-10-21 09:38:52,552 INFO TX: 2004; RX: 2003; Est. Dropped: 1; Est. Drop rate: 0.
↪0499%
```

In another bash window, you should connect to the TRex console doing :

```
cd /opt/trex/vX.XX/ #use completion here to find your corresponding TRex version
./trex-console -r
capture record start --rx [port number] --limit 10000
```

Check on the TRex window that the following log appears just after capture is started:

```
Starting packet capturing up to 10000 packets               [SUCCESS]
*** Capturing ID is set to '8' ***
*** Please call 'capture record stop --id 8 -o <out.pcap>' when done ***
```

Then **before end of traffic generation**, stop capture and save it as a PCAP file:

```
capture record stop --id 8 -o /tmp/nfvb/record.pcap
```

Note: Provide a shared path with between NFVbench container and your host to retrieve pcap file

Check on the TRex window that the following log appears just after capture is started:

```
Stopping packet capture 8                                   [SUCCESS]
Writing up to 10000 packets to '/tmp/nfvb/record.pcap'      [SUCCESS]
Removing PCAP capture 8 from server                         [SUCCESS]
```

User info data

The `--user-info` option allows you to pass custom information as a JSON string. This information will be available through JSON output and also exported to `fluentd` and can be used in results post-processing.

Example of use :

```
nfvbench ``--user-info='{"status":"explore","description":{"target":"lab","ok":true,
↪"version":2020}'``
```

Note: only JSON string is allowed

`--user-info` can be used for determining theoretical max rate. In some cases, an overhead encapsulation exists between NFVbench and SUT so NFVbench will not reach line rate inside SUT due to this extra encapsulation. To calculate this theoretical line rate inside SUT, NFVbench will use a reserved key: `extra_encapsulation_bytes` in `--user-info` property.

```
nfvbench ``--user-info='{"extra_encapsulation_bytes": 28}'``
```

As a result, NFVbench will return two values `theoretical_tx_rate_bps` and `theoretical_tx_rate_pps`:

```
"ndr": {
    "duration_sec": 2.0,
    "initial_rate_type": "rate_percent",
    "l2frame_size": "64",
    "load_percent_per_direction": 100.0,
    "rate_bps": 20000000000.0,
    "rate_percent": 200.0,
    "rate_pps": 29761904,
```

(continues on next page)

(continued from previous page)

```
    "stats": {
        ...
        "offered_tx_rate_bps": 15000000000.0,
        ...
        "theoretical_tx_rate_bps": 15000000000.0,
        "theoretical_tx_rate_pps": 22321428.57142857,
        "total_tx_rate": 22321428
    },
```

In the above example, line rate is 20Gbps but NFVbench is outside SUT and a SDN gateway add an extra encapsulation of 28 bytes. Overall, theoretical line rate inside SUT is only 15 Gbps for 64 bytes packet size and it will be this max capacity treated by the target compute node.

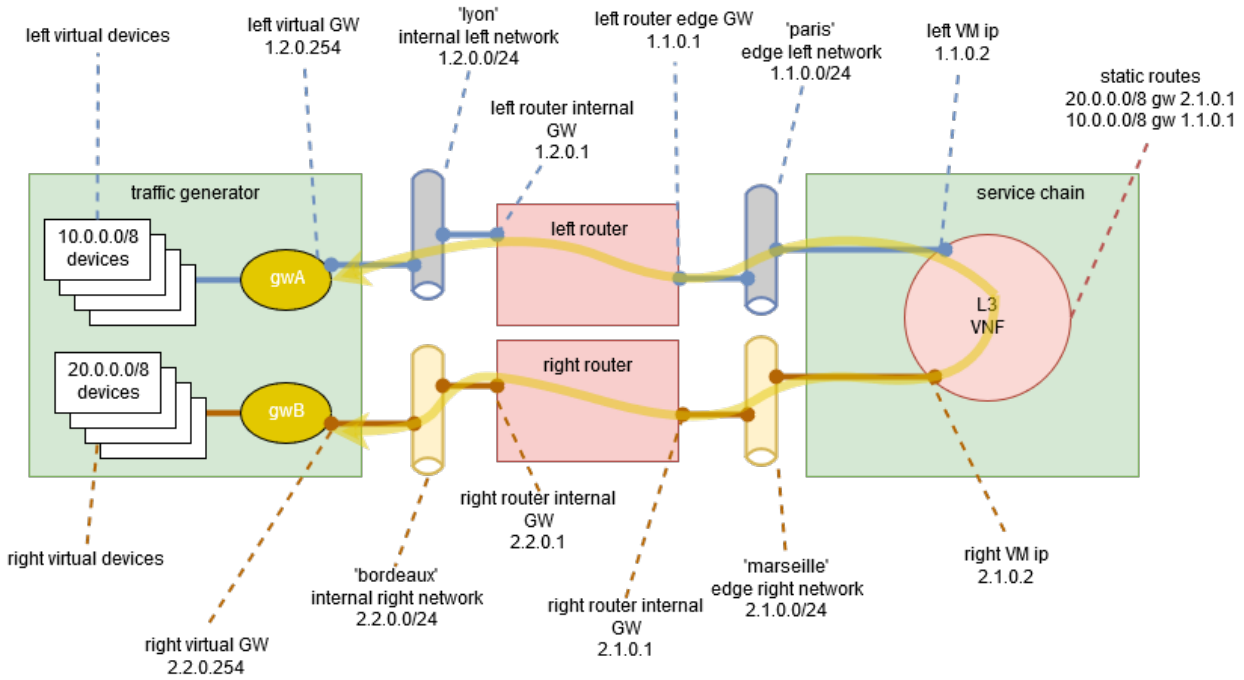
7.1.6 PVP L3 Router Internal Chain

NFVbench can measure the performance of 1 L3 service chain that are setup by NFVbench (VMs, routers and networks).

PVP L3 router chain is made of 1 VNF (in vpp mode) and has exactly 2 end network interfaces (left and right internal network interfaces) that are connected to 2 neutron routers with 2 edge networks (left and right edge networks). The PVP L3 router service chain can route L3 packets properly between the left and right networks.

To run NFVbench on such PVP L3 router service chain:

- explicitly tell NFVbench to use PVP service chain with L3 router option by adding `-l3` or `--l3-router` to NFVbench CLI options or `l3_router: true` in config
- explicitly tell NFVbench to use VPP forwarder with `vm_forwarder: vpp` in config
- **specify the 2 end point networks (networks between NFVBench and neutron routers) of your environment in `internal_`**
  - The two networks specified will be created if not existing in Neutron and will be used as the end point networks by NFVbench ('lyon' and 'bordeaux' in the diagram below)
- **specify the 2 edge networks (networks between neutron routers and loopback VM) of your environment in `edge_networ`**
  - The two networks specified will be created if not existing in Neutron and will be used as the router gateway networks by NFVbench ('paris' and 'marseille' in the diagram below)
- specify the router gateway IPs for the PVPL3 router service chain (1.2.0.1 and 2.2.0.1)
- specify the traffic generator gateway IPs for the PVPL3 router service chain (1.2.0.254 and 2.2.0.254 in diagram below)
- specify the packet source and destination IPs for the virtual devices that are simulated (10.0.0.0/8 and 20.0.0.0/8)



nfvbench configuration file:

```
vm_forwarder: vpp

traffic_generator:
    ip_addrs: ['10.0.0.0/8', '20.0.0.0/8']
    tg_gateway_ip_addrs: ['1.2.0.254', '2.2.0.254']
    gateway_ip_addrs: ['1.2.0.1', '2.2.0.1']

internal_networks:
    left:
        name: 'lyon'
        cidr: '1.2.0.0/24'
        gateway: '1.2.0.1'
    right:
        name: 'bordeaux'
        cidr: '2.2.0.0/24'
        gateway: '2.2.0.1'

edge_networks:
    left:
        name: 'paris'
        cidr: '1.1.0.0/24'
        gateway: '1.1.0.1'
    right:
        name: 'marseille'
        cidr: '2.1.0.0/24'
        gateway: '2.1.0.1'
```

Upon start, NFVbench will: - first retrieve the properties of the left and right networks using Neutron APIs, - extract the underlying network ID (typically VLAN segmentation ID), - generate packets with the proper VLAN ID and measure traffic.

Please note: `l3_router` option is also compatible with external routers. In this case NFVBench will use `EXT` chain.

Note: Using a long NFVbench run test, end-to-end connectivity can be lost depending on ARP stale time SUT configuration.

To avoid this issue, activate Gratuitous ARP stream using `--gratuitous-arp` or `-garp` option.

7.1.7 NFVbench Xtesting test cases and Xtesting CI integration

NFVbench can leverages on Xtesting CI and the common test case execution proposed by Xtesting. Thanks to a simple test case list, this tool deploys anywhere plug-and-play CI/CD toolchains in a few commands. In addition, it supports multiple components such as Jenkins and Gitlab CI (test schedulers) and multiple deployment models such as all-in-one or centralized services.

NFVbench using Xtesting and Xtesting CI will permit:

- smoothly assemble multiple heterogeneous test cases
- generate the Jenkins jobs
- deploy local CI/CD toolchains everywhere
- dump all test case results and logs for third-party conformance review

Xtesting CI only requires GNU/Linux as Operating System and asks for a few dependencies as described in Deploy your own Xtesting CI/CD toolchains:

- python-virtualenv
- docker.io
- git

Please note the next two points depending on the GNU/Linux distributions and the network settings:

- SELinux: you may have to add –system-site-packages when creating the virtualenv ("Aborting, target uses selinux but python bindings (libselinux-python) aren't installed!")
- Proxy: you may set your proxy in env for Ansible and in systemd for Docker https://docs.docker.com/config/daemon/systemd/#httphttps-proxy

Here is the default NFVbench tree on the host running nfvbench container as proposed in `xtesting/ansible/host_vars/127.0.0.1` file:

- `/home/opnfv/nfvbench/config`: contains nfvbench config files including `nfvbench.cfg`, the default config file used by xtesting test cases
- `/home/opnfv/nfvbench/results`: top directory to write nfvbench results and log files

File content:

```
docker_args:
  env: {}
  params:
    net: host
    privileged: true
  volumes:
    - /lib/modules/$(uname -r):/lib/modules/$(uname -r)
    - /usr/src/kernels:/usr/src/kernels -v /dev:/dev
    - /home/opnfv/nfvbench/config:/etc/nfvbench
    - /home/opnfv/nfvbench/results:/var/lib/xtesting/results
```

Please note: if you want to use a different directory structure on the host, replace `/home/opnfv/nfvbench/*` paths with appropriate paths to permit NFVbench container to access config file and results directory. The config and results paths do not need to share the same root directory.

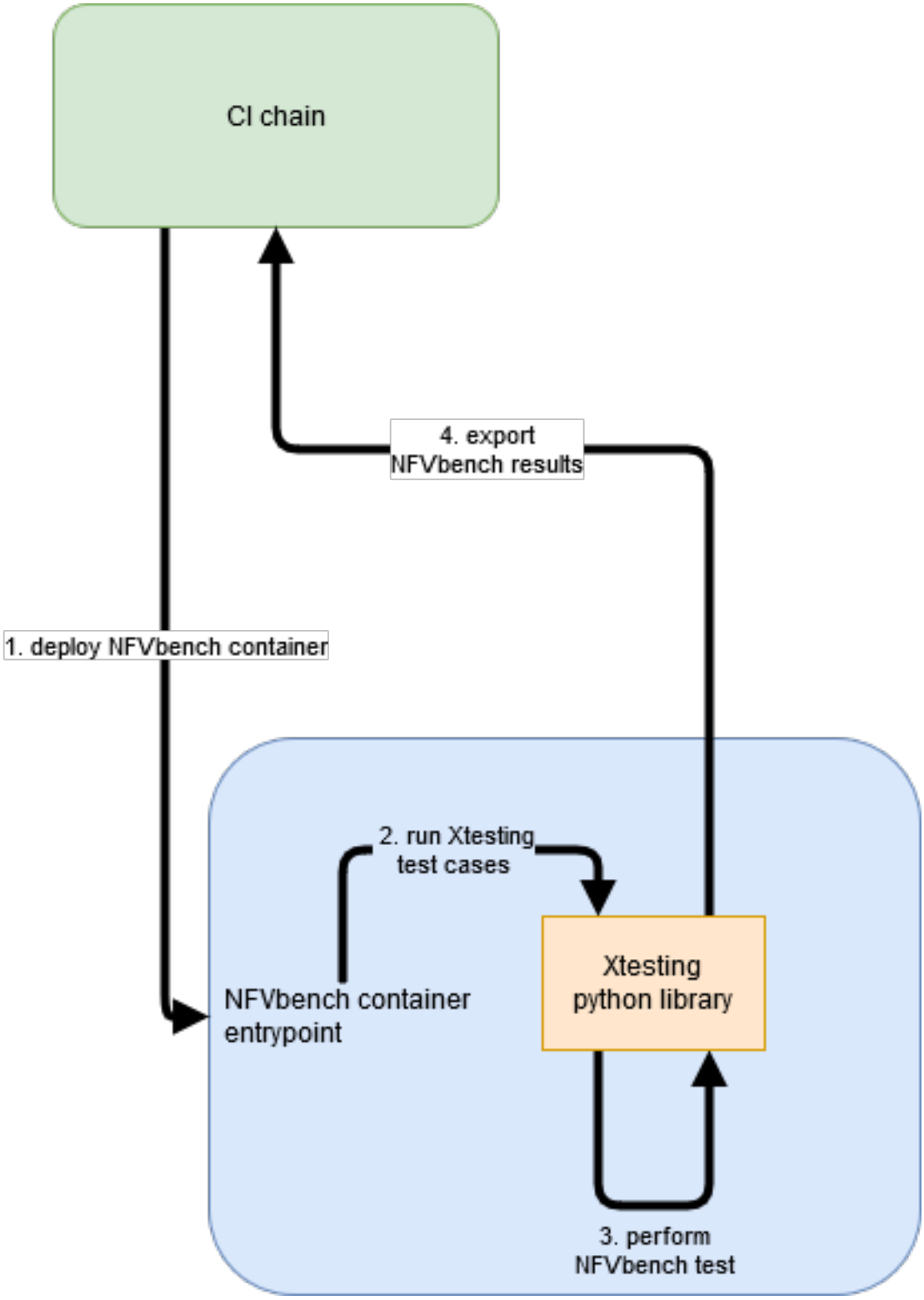
To deploy your own CI toolchain running NFVbench:

```
virtualenv nfvbench
. nfvbench/bin/activate
pip install ansible
ansible-galaxy install collivier.xtesting
git clone https://gerrit.opnfv.org/gerrit/nfvbench nfvbench
ansible-playbook nfvbench/xtesting/ansible/site.yml
```

7.1.8 NFVbench Xtesting test cases and existing CI chain

For test automation purpose, Xtesting framework can be used as an executor of NFVbench test cases and called by a CI chain (Jenkins, Gitlab CI . . .). Xtesting use a testcases.yaml file to list and run test case. One basic testcases.yaml is provided by NFVbench natively but can be override.

Example of CI scenario:



1. Run NFVbench container using Xtesting python library

The NFVbench container can be started using docker run command.

To run NFVbench using docker run:

```
docker run --rm \
    -e S3_ENDPOINT_URL=http://127.0.0.1:9000 \
    -e S3_DST_URL=s3://xtesting/$BUILD_TAG/$JOB_NAME-$BUILD_ID \
    -e HTTP_DST_URL=http://127.0.0.1:8181/$BUILD_TAG/$JOB_NAME-$BUILD_ID \
    -e AWS_ACCESS_KEY_ID=xtesting \
    -e AWS_SECRET_ACCESS_KEY=xtesting \
    -e TEST_DB_URL=http://127.0.0.1:8000/api/v1/results \
    -e NODE_NAME=nfvbench \
    -e BUILD_TAG=$BUILD_TAG \
    --privileged \
    -v /lib/modules/$(uname -r):/lib/modules/$(uname -r) \
    -v /usr/src/kernels:/usr/src/kernels -v /dev:/dev \
    -v $HOME/nfvbench/config:/etc/nfvbench \
    -v $HOME/workspace/$JOB_NAME/results:/var/lib/xtesting/results \
    opnfv/nfvbench run_tests -t 10kpps-pvp-run -r -p
```

Docker options	Description
–rm	clean up container after execution
-e S3_ENDPOINT_URL	(Xtesting) Environnement variable used to store NFVbench artifacts to Minio
-e S3_DST_URL	(Xtesting) Environnement variable used for S3 storage destination
-e HTTP_DST_URL	(Xtesting) Environnement variable used for S3www service
-e AWS_ACCESS_KEY_ID	(Xtesting) Environnement variable used for S3 access key
-e AWS_SECRET_ACCESS_KEY	(Xtesting) Environnement variable used for S3 access secret
-e TEST_DB_URL	(Xtesting) Environnement variable used to export NFVbench results in DB
-e NODE_NAME	(Xtesting) Environnement variable used as result key identifier in DB
-e BUILD_TAG	(Xtesting) Environnement variable used as result key identifier in DB
–privileged	(optional) required if SELinux is enabled on the host
-v /lib/modules:/lib/modules	needed by kernel modules in the container
-v /usr/src/kernels:/usr/src/kernels	needed by TRex to build kernel modules when needed
-v /dev:/dev	needed by kernel modules in the container
-v $HOME/nfvbench/config:/etc/nfvbench	folder mapping to pass config files between the host and the docker space (see example above). Here we map the $HOME/nfvbench/config directory on the host to the /etc/nfvbench directory in the container. Any other mapping can work as well
-v $HOME/workspace/$JOB_NAME/results:/var/lib/xtesting/results	(Xtesting) folder mapping to pass files between the CI chain workspace and the docker space/results Xtesting folder. It is used to orchestrator (Jenkins, Gitlab . . .)
opnfv/nfvbench	container image name
run_tests	(Xtesting) Xtesting command to run test cases
-t 10kpps-pvp-run	(Xtesting) Xtesting parameter: Test case or tier (group of tests) to be executed. It will run all the test if not specified.
-r	(Xtesting) Xtesting parameter: publish result to database
-p	(Xtesting) Xtesting parameter: publish artifacts to a S3 service

2. Run Xtesting test cases

Executed directly by NFVbench docker entrypoint after docker start.

3. Perform NFVbench test

Xtesting call NFVbench python script to execute test case scenario and wait for run to be terminated.

4. Export NFVbench result

If `-r` option is used, results are pushed to a DB through Xtesting. If `-p` option is used, results are pushed to a S3 service through Xtesting.

Override testcases.yaml file

To replace existing testcases.yaml file, using Xtesting CI add the volume mapping in `xtesting/ansible/host_vars/127.0.0.1` file:

```
docker_args:
env: {}
volumes:
  - /lib/modules/$(uname -r):/lib/modules/$(uname -r)
  - /usr/src/kernels:/usr/src/kernels -v /dev:/dev
  - /home/opnfv/nfvbench/config:/etc/nfvbench
  - /home/opnfv/nfvbench/results:/var/lib/xtesting/results
  - /home/opnfv/nfvbench/xtesting/testcases.yaml:/usr/local/lib/python3.6/dist-
↪packages/xtesting/ci/testcases.yaml
```

- `/home/opnfv/nfvbench/xtesting/testcases.yaml:/usr/local/lib/python3.6/dist-packages/xtesting/ci/testcases.yaml` : volume mapping to pass testcases.yaml file between the host and the docker space. Host path required testcases.yaml file inside.

To replace existing testcases.yaml file, using NFVbench container:

```
docker run --name nfvbench --detach --privileged \
    -v /lib/modules/$(uname -r):/lib/modules/$(uname -r) \
    -v /usr/src/kernels:/usr/src/kernels \
    -v /dev:/dev \
    -v $HOME/nfvbench/config:/etc/nfvbench \
    -v $HOME/nfvbench/results:/var/lib/xtesting/results \
    -v $HOME/nfvbench/xtesting/testcases.yaml:/usr/local/lib/python3.8/dist-packages/
↪xtesting/ci/testcases.yaml \
    opnfv/nfvbench
```

- `$HOME/nfvbench/xtesting/testcases.yaml:/usr/local/lib/python3.8/dist-packages/xtesting/ci/testcases.yaml` : volume mapping to pass testcases.yaml file between the host and the docker space. Host path required testcases.yaml file inside.

Example of Xtesting test case

```
---
tiers:
    -
        name: nfvbench
        order: 1
        description: 'Data Plane Performance Testing'
        testcases:
            -
                case_name: 10kpps-pvp-run
                project_name: nfvbench
                criteria: 100
                blocking: true
                clean_flag: false
                description: ''
                run:
```

(continues on next page)

(continued from previous page)

```
                    name: 'bashfeature'
                    args:
                        cmd:
                            - nfvbench -c /etc/nfvbench/nfvbench.cfg --rate 10kpps
```

Examples of manual run

If NFVbench container is already started in CLI mode (see Starting NFVbench in CLI mode dedicated chapter). To do a single run at 10,000pps bi-directional (or 5kpps in each direction) using the PVP packet path:

```
docker exec -it nfvbench run_tests -t 10kpps-pvp-run
```

Xtesting option used:

- `-t 10kpps-pvp-run` : specify the test case to run

To pass all test cases:

```
docker exec -it nfvbench run_tests -t all
```

Xtesting option used:

- `-t all` : select all test cases existing in testcases.yaml file

7.1.9 MPLS encapsulation feature

This feature allows to generate packets with standard MPLS L2VPN double stack MPLS labels, where the outer label is transport and the inner label is VPN. The top layer of a packets encapsulated inside MPLS L2VPN seems to be an Ethernet layer with the rest of the IP stack inside. Please refer to RFC-3031 for more details. The whole MPLS packet structure looks like the following:

###[Ethernet]### dst = ['00:8a:96:bb:14:28'] src = 3c:fd:fe:a3:48:7c type = 0x8847

###[MPLS]### <———— Outer Label label = 16303 cos = 1 s = 0 ttl = 255

###[MPLS]### <———— Inner Label label = 5010 cos = 1 s = 1 ttl = 255

###[Ethernet]### dst = fa:16:3e:bd:02:b5 src = 3c:fd:fe:a3:48:7c type = 0x800

###[IP]### version = 4 ihl = None tos = 0x0 len = None id = 1 flags = frag = 0 ttl = 64 proto = udp chksum = None src = 16.0.0.1 dst = 48.0.0.1 options

###[UDP]### sport = 53 dport = 53 len = None chksum = None

Example: nfvbench generates mpls traffic port A —-> port B. This example assumes openstack is at the other end of the mpls tunnels. Packets generated and sent to port B are delivered to the MPLS domain infrastructure which will transport that packet to the other end of the MPLS transport tunnel using the outer label. At that point, the outer label is decapsulated and the inner label is used to select the destination openstack network. After decapsulation of the inner label, the resulting L2 frame is then forwarded to the destination VM corresponding to the destination MAC. When the VM receives the packet, it is sent back to far end port of the traffic generator (port B) using either L2 forwarding or L3 routing though the peer virtual interface. The return packet is then encapsulated with the inner label first then outer label to reach nfvbench on port B.

Only 2 MPLS labels stack is supported. If more than two labels stack is required then these operations should be handled by MPLS transport domain where nfvbench is attached next-hop mpls router and rest of the mpls domain should be configured accordingly to be able pop/swap/push labels and deliver packet to the proper destination based

on an initial transport label injected by nfvbench, VPN label should stay unchanged until its delivered to PE (compute node). Set nfvbench 'mpls' parameter to 'true' to enable MPLS encapsulation. When this option is enabled internal networks 'network type' parameter value should be 'mpls' MPLS and VxLAN encapsulations are mutual exclusive features if 'mpls' is 'true' then 'vxlan' should be set to 'false' and vise versa. no_flow_stats, no_latency_stats, no_latency_streams parameters should be set to 'true' because these features are not supported at the moment. In future when these features will be supported they will require special NIC hardware.

Example of 1-chain MPLS configuration:

internal_networks:

left: network_type: mpls segmentation_id: 5010 mpls_transport_labels: 16303 physical_network: phys_sriov0

right: network_type: mpls segmentation_id: 5011 mpls_transport_labels: 16303 physical_network: phys_sriov1

Example of 2-chain MPLS configuration:

internal_networks:

left: network_type: mpls segmentation_id: [5010, 5020] mpls_transport_labels: [16303, 16304] physical_network: phys_sriov0

right: network_type: mpls segmentation_id: [5011, 5021] mpls_transport_labels: [16303, 16304] physical_network: phys_sriov1

Example of how to run: nfvbench –rate 50000pps –duration 30 –mpls

7.1.10 External Chains

NFVbench can measure the performance of 1 or more L3 service chains that are setup externally using OpenStack or without OpenStack. Instead of being setup by NFVbench, the complete environment (VNFs and networks) must be setup prior to running NFVbench.

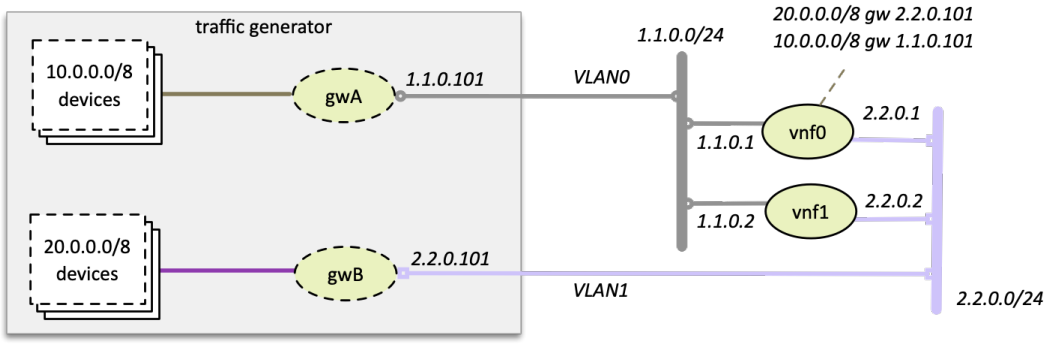
Each external chain is made of 1 or more VNFs and has exactly 2 edge network interfaces (left and right network interfaces) that are connected to 2 edge networks (left and right networks). The 2 edge networks for each chain can either be shared across all chains or can be independent.

The internal composition of a multi-VNF service chain can be arbitrary (usually linear) as far as NFVbench is concerned, the only requirement is that the service chain can route L3 packets properly between the left and right networks.

The network topology of the service chains is defined by the "service_chain_shared_net" option in the NFVbench configuration file.

Shared Edge Networks

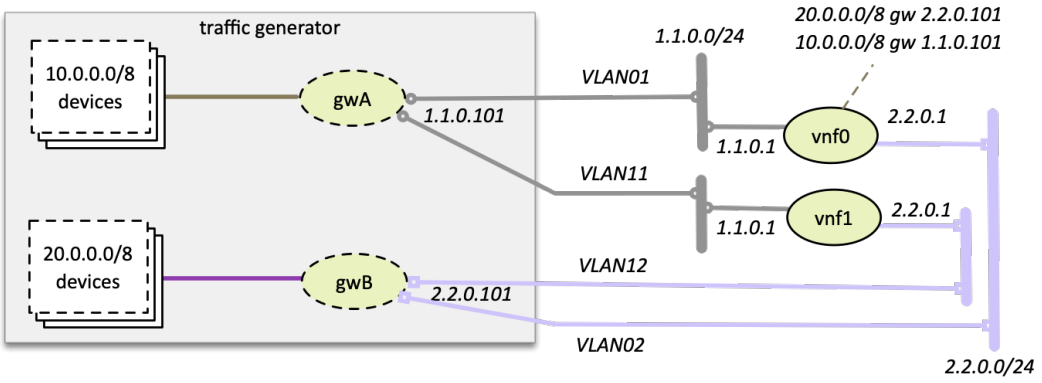
This option is defined when "service_chain_shared_net" is set to true. All chains must share the same 2 edge networks and the VNF gateway IP addresses on each edge must all belong to the same subnet.



The main advantage of this mode is that only 2 network segments are needed to support an arbitrary number of chains.

Multi-VLAN Edge Networks

This option is defined when "service_chain_shared_net" is set to false (default). Each chain has its own dedicated left and right network and there is no inter-chain constraint on the VNF IP addresses since they all belong to different network segments.

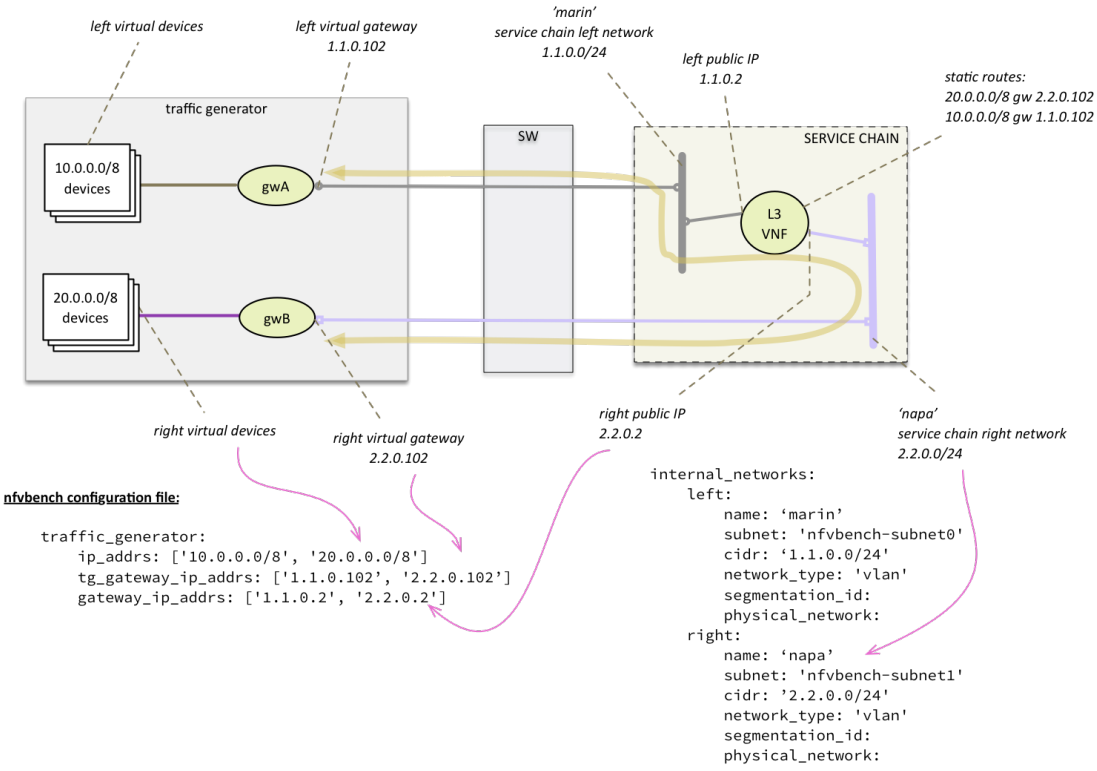


The advantage of this mode is that the configuration of the VNFs can be made identical (same gateway IP addresses, same static routes). However this mode requires 2 network segments per chain.

Detailed Example

To run NFVbench on an external service chains using shared edge networks:

- tell NFVbench to use external service chain by adding "-sc EXT" or "–service-chain EXT" to NFVbench CLI options
- specify the number of external chains using the "-scc" option (defaults to 1 chain)
- **if OpenStack is used:**
  - specify the name of the 2 edge networks in "external_networks" in the NFVbench configuration file
  - The two networks specified have to exist in Neutron ('napa' and 'marin' in the diagram below)
- **if OpenStack is not used:**
  - specify the VLAN id to use for the 2 edge networks in "vlans" in the NFVbench configuration file
- specify the VNF gateway IPs for the external service chains (1.1.0.2 and 2.2.0.2)
- specify the traffic generator gateway IPs for the external service chains (1.1.0.102 and 2.2.0.102 in diagram below)
- specify the packet source and destination IPs for the virtual devices that are simulated (10.0.0.0/8 and 20.0.0.0/8)



L3 routing must be enabled in the VNF and configured to:

- reply to ARP requests to its public IP addresses on both left and right networks

- route packets from each set of remote devices toward the appropriate dest gateway IP in the traffic generator using 2 static routes (as illustrated in the diagram)

Upon start, NFVbench will: - first retrieve the properties of the left and right networks using Neutron APIs, - extract the underlying network ID (typically VLAN segmentation ID), - generate packets with the proper VLAN ID and measure traffic.

Note that in the case of multiple chains, all chains end interfaces must be connected to the same two left and right networks. The traffic will be load balanced across the corresponding gateway IP of these external service chains.

7.1.11 NFVbench Fluentd Integration

NFVbench has an optional fluentd integration to save logs and results.

Configuring Fluentd to receive NFVbench logs and results

The following configurations should be added to Fluentd configuration file to enable logs or results.

To receive logs, and forward to a storage server:

In the example below nfvbench is the tag name for logs (which should be matched with logging_tag under NFVbench configuration), and storage backend is elasticsearch which is running at localhost:9200.

```
<match nfvbench.**>
@type copy
<store>
    @type elasticsearch
    host localhost
    port 9200
    logstash_format true
    logstash_prefix nfvbench
    utc_index false
    flush_interval 15s
</store>
</match>
```

To receive results, and forward to a storage server:

In the example below resultnfvbench is the tag name for results (which should be matched with result_tag under NFVbench configuration), and storage backend is elasticsearch which is running at localhost:9200.

```
<match resultnfvbench.**>
@type copy
<store>
    @type elasticsearch
    host localhost
    port 9200
    logstash_format true
    logstash_prefix resultnfvbench
    utc_index false
    flush_interval 15s
</store>
</match>
```

Configuring NFVbench to connect Fluentd

To configure NFVbench to connect Fluentd, fill following configuration parameters in the configuration file

Configuration	Description
logging_tag	Tag for NFVbench logs, it should be the same tag defined in Fluentd configuration
result_tag	Tag for NFVbench results, it should be the same tag defined in Fluentd configuration
ip	ip address of Fluentd server
port	port number of Fluentd serverd

An example of configuration for Fluentd working at 127.0.0.1:24224 and tags for logging is nfvbench and result is resultnfvbench

```
fluentd:
    # by default (logging_tag is empty) nfvbench log messages are not sent to fluentd
    # to enable logging to fluents, specify a valid fluentd tag name to be used for␣
↪the
    # log records
    logging_tag: nfvbench

    # by default (result_tag is empty) nfvbench results are not sent to fluentd
    # to enable sending nfvbench results to fluentd, specify a valid fluentd tag name
    # to be used for the results records, which is different than logging_tag
    result_tag: resultnfvbench

    # IP address of the server, defaults to loopback
    ip: 127.0.0.1

    # port # to use, by default, use the default fluentd forward port
    port: 24224
```

Example of logs and results

An example of log obtained from fluentd by elasticsearch:

```
{
  "_index": "nfvbench-2017.10.17",
  "_type": "fluentd",
  "_id": "AV8rhnCjTgGF_dX8DiKK",
  "_version": 1,
  "_score": 3,
  "_source": {
    "loglevel": "INFO",
    "message": "Service chain 'PVP' run completed.",
    "@timestamp": "2017-10-17T18:09:09.516897+0000",
    "runlogdate": "2017-10-17T18:08:51.851253+0000"
  },
  "fields": {
    "@timestamp": [
      1508263749516
    ]
  }
}
```

For each packet size and rate a result record is sent. Users can label those results by passing –user-label parameter to NFVbench run

And the results of this command obtained from fluentd by elasticsearch:


```
{
  "_index": "resultnfvbench-2017.10.17",
  "_type": "fluentd",
  "_id": "AV8rjYlbTgGF_dX8Drl1",
  "_version": 1,
  "_score": null,
  "_source": {
    "compute_nodes": [
      "nova:compute-3"
    ],
    "total_orig_rate_bps": 200000000,
    "@timestamp": "2017-10-17T18:16:43.755240+0000",
    "frame_size": "64",
    "forward_orig_rate_pps": 148809,
    "flow_count": 10000,
    "avg_delay_usec": 6271,
    "total_tx_rate_pps": 283169,
    "total_tx_rate_bps": 190289668,
    "forward_tx_rate_bps": 95143832,
    "reverse_tx_rate_bps": 95145836,
    "forward_tx_rate_pps": 141583,
    "chain_analysis_duration": "60.091",
    "service_chain": "PVP",
    "version": "1.0.10.dev1",
    "runlogdate": "2017-10-17T18:10:12.134260+0000",
    "Encapsulation": "VLAN",
    "user_label": "nfvbench-label",
    "min_delay_usec": 70,
    "profile": "traffic_profile_64B",
    "reverse_rx_rate_pps": 68479,
    "reverse_rx_rate_bps": 46018044,
    "reverse_orig_rate_pps": 148809,
    "total_rx_rate_bps": 92030085,
    "drop_rate_percent": 51.6368455626846,
    "forward_orig_rate_bps": 100000000,
    "bidirectional": true,
    "vSwitch": "OPENVSWITCH",
    "sc_count": 1,
    "total_orig_rate_pps": 297618,
    "type": "single_run",
    "reverse_orig_rate_bps": 100000000,
    "total_rx_rate_pps": 136949,
    "max_delay_usec": 106850,
    "forward_rx_rate_pps": 68470,
    "forward_rx_rate_bps": 46012041,
    "reverse_tx_rate_pps": 141586
  },
  "fields": {
    "@timestamp": [
      1508264203755
    ]
  },
  "sort": [
    1508264203755
  ]
}
```

7.1.12 NFVbench Kibana visualization: overview

The fluentd integration offers the possibility to use elasticsearch and kibana as a visualization chain.

Chain overview:

Example of NFVbench visualizations

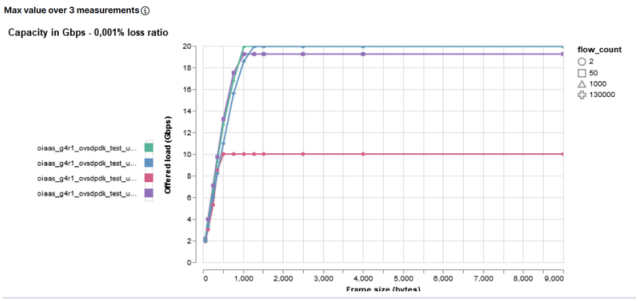
Kibana offers a lot of visualization type (line and bar charts, pie, time series chart, data table ...) and also provide a plugin to develop graph using Vega. In the below examples, visualizations are based on an NDR result and are developed using Vega-lite. Data are aggregated using `user_label` and `flow_count` properties.

In `kibana/visualizations/` pre-created graphs are available into json files.

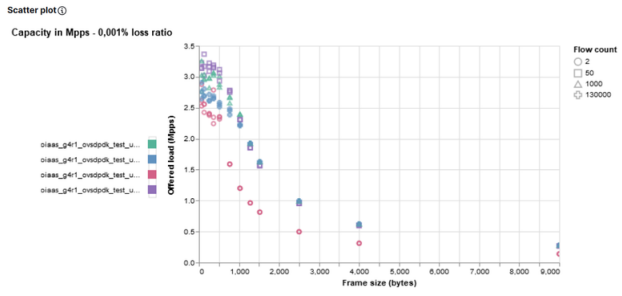
For NDR capacity in Gbps using line chart, the offered load in Gbps (`offered_tx_rate_bps`) is used and only the maximum value of the aggregation is kept. For NDR capacity in Mpps using line chart, the actual TX rate is used (`rate_pps`) and only the maximum value of the aggregation is kept.

Scatter plot graphs use the same values but keep all values instead of keeping maximum.

Example of a line chart:

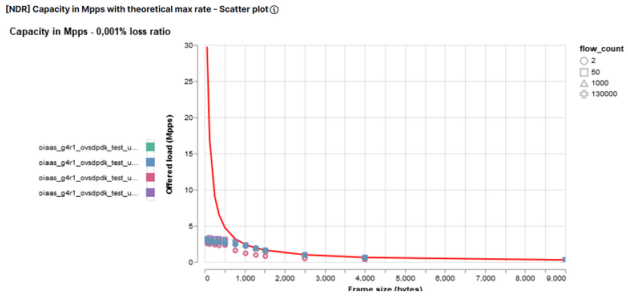


Example of a scatter plot chart:



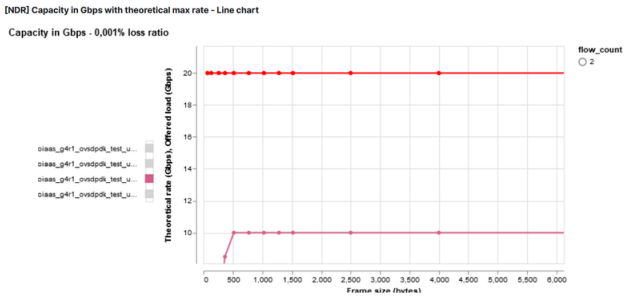
Vega offers the possibility to add another graph as a new layer of current graph. This solution is used to combine NFVbench results and theoretical line rate. Using `extra_encapsulation_bytes` in –user-info property (see User info data section), the theoretical max value (for bps and pps) will be calculated and can be used in graph through `theoretical_tx_rate_bps` and `theoretical_tx_rate_pps` properties.

Example of chart with theoretical value (red line):



Each Vega graph can be moved, zoomed (using mouse scroll) and one set of data can be selected.

Example:



These visualizations are included into Kibana dashboard for a synthesis of one set of result (i.e. same `user_label` value) or for comparison (i.e. a selection of `user_label` values). See *Kibana user guide: Filter dashboards and visualizations* for more details about `user_label` selection.

All these visualizations and dashboards are saved into the `export.ndjson` file and can be imported in an existing Kibana. See *Import Kibana dashboards and visualization*.

Import Kibana dashboards and visualization

To import Kibana dashboard and visualization:

```
curl -X POST localhost:5601/api/saved_objects/_import -H "kbn-xsrf: true" --form␣
↪file=@export.ndjson
```

Note: `.kibana` index must exists in elasticsearch.

Note: `.kibana` index is created automatically after a first deployment and configuration of Kibana.

7.1.13 Kibana user guide: Filter dashboards and visualizations

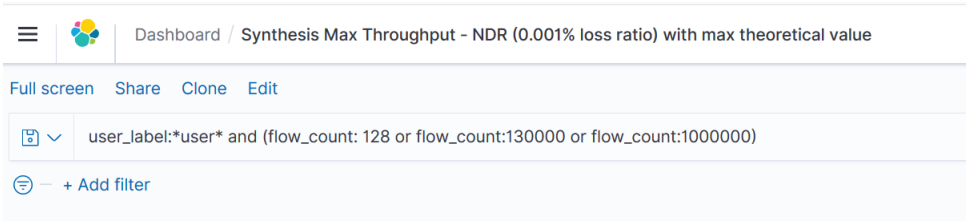
Filter Kibana dashboard or visualization using Kibana query language (KQL)

```
user_label:*demo* and (flow_count: 128 or flow_count:130000 or flow_count:1000000)
```

Note: This query will filter all user label which contains `demo` in the value and filter 3 flow count (128, 130k, 1M).

Note: `flow_count` is a number so KQL query can not contain formatted string.

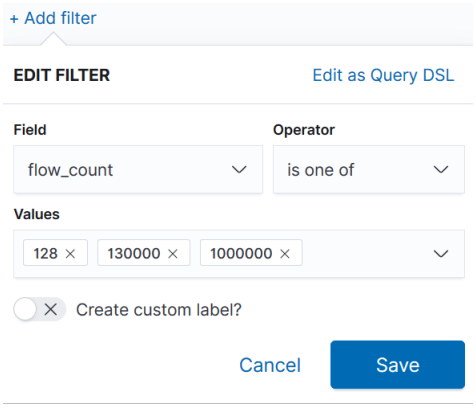
Example:



Filter Kibana dashboard or visualization using Kibana filters

Kibana offers the possibility to add filter by selecting field and operator (is, is not, is one of, is not one of, exists, does not exist).

Example:



7.1.14 Testing SR-IOV

NFVbench supports SR-IOV with the PVP and PVVP packet flow. Most use cases for SR-IOV only require single VNF chains (NxPVP). Daisy chaining VNFs with SR-IOV (PVVP) requires selecting either SR-IOV for the middle network or a fast vswitch (using the standard OVS for that purpose works but would be a serious bottleneck)

Instructions below refer to the PVP or PVVP use cases. For external chains using SR-IOV, select the VLAN tagging option that corresponds to the external chains SR-IOV setting.

Pre-requisites

To test SR-IOV you need to have compute nodes configured to support one or more SR-IOV interfaces (also knows as PF or physical function) and you need OpenStack to be configured to support SR-IOV. You will also need to know: - the name of the physical networks associated to the SR-IOV interfaces (this is a configuration in Nova compute) - the VLAN range that can be used on the switch ports that are wired to the SR-IOV ports. Such switch ports are normally configured in trunk mode with a range of VLAN ids enabled on that port

For example, in the case of 2 SR-IOV ports per compute node, 2 physical networks are generally configured in OpenStack with a distinct name. The VLAN range to use is is also allocated and reserved by the network administrator and in coordination with the corresponding top of rack switch port configuration.

Configuration

To enable SR-IOV test, you will need to provide the following configuration options to NFVbench (in the configuration file). This example instructs NFVbench to create the left and right networks of a PVP packet flow to run on 2 SRIOV ports named "phys_sriov0" and "phys_sriov1" using resp. segmentation_id 2000 and 2001:

```
sriov: true
internal_networks:
   left:
       segmentation_id: 2000
       physical_network: phys_sriov0
   right:
       segmentation_id: 2001
       physical_network: phys_sriov1
```

The segmentation ID fields must be different. In the case of PVVP, the middle network also needs to be provisioned properly. The same physical network can also be shared by the virtual networks but with different segmentation IDs.

Multi-Chaining

The above configuration works for multi-chaining and shared network ("service_chain_shared_net" set to true). In that case all VNFs will share the same left and right network/VLAN.

In the case of non shared network ("service_chain_shared_net" set to false), the segmentation_id fields must contain a list of distinct VLANs to use for each chain. Example of configuration for 3 chains:

```
sriov: true
internal_networks:
   left:
       segmentation_id: [2000, 2001, 2002]
       physical_network: phys_sriov0
   right:
       segmentation_id: [2100, 2101, 2102]
       physical_network: phys_sriov1
```

Alternatively it is also possible to specify different physnets per chain:

```
sriov: true
internal_networks:
   left:
       segmentation_id: [2000, 2001, 2002]
       physical_network: [phys_sriov0, phys_sriov2, phys_sriov4]
   right:
```

(continues on next page)

(continued from previous page)

```
      segmentation_id: [2100, 2101, 2102]
      physical_network: [phys_sriov1, phys_srviov3, phys_sriov5]
```

NFVbench cores with SR-IOV

The default core count for NFVbench/TRex may not be sufficient for higher throughput line cards (greater than 10Gbps). This will result in warning messages such as:

```
INFO WARNING: There is a significant difference between requested TX rate (119047618)␣
↪and actual TX rate (38897379).
The traffic generator may not have sufficient CPU to achieve the requested TX rate.
```

In that case it is recommended to try allocating more cores to TRex using the cores property in the configuration file, for example to set to 8 cores:

```
cores: 8
```

It is also advisable to increase the number of vcpus in the VMs:

VM Flavor for SR-IOV and NIC NUMA socket placement

Because SR-IOV throughput uses a lot of CPU in the VM, it is recommended to increase the vcpu count, for example to 4 vcpus:

```
flavor:
  # Number of vCPUs for the flavor
  vcpus: 4
  # Memory for the flavor in MB
  ram: 8192
  # Size of local disk in GB
  disk: 0
  extra_specs:
      "hw:cpu_policy": dedicated
```

If the 2 selected ports reside on NICs that are on different NUMA sockets, you will need to explicitly tell Nova to use 2 numa nodes in the flavor used for the VMs in order to satisfy the filters, for example:

```
flavor:
  # Number of vCPUs for the flavor
  vcpus: 4
  # Memory for the flavor in MB
  ram: 8192
  # Size of local disk in GB
  disk: 0
  extra_specs:
      "hw:cpu_policy": dedicated
      "hw:numa_nodes": 2
```

Failure to do so might cause the VM creation to fail with the Nova error “Instance creation error: Insufficient compute resources: Requested instance NUMA topology together with requested PCI devices cannot fit the given host NUMA topology.”

Example of configuration file (shared network)

Single chain or multi-chain with shared network (only requires 2 segmentation ID for all chains):

```
flavor:
   # Number of vCPUs for the flavor
   vcpus: 4
   # Memory for the flavor in MB
   ram: 8192
   # Size of local disk in GB
   disk: 0
   extra_specs:
      "hw:cpu_policy": dedicated
cores: 8
sriov: true
internal_networks:
   left:
      segmentation_id: 3830
      physical_network: phys_sriov0
   right:
      segmentation_id: 3831
      physical_network: phys_sriov1
```

Example of full run 2xPVP shared network SR-IOV:

```
2018-12-03 18:24:07,419 INFO Loading configuration file: /tmp/nfvbench/sriov.yaml
2018-12-03 18:24:07,423 INFO -c /tmp/nfvbench/sriov.yaml --rate 10Mpps --duration 1 -
↪scc 2 --no-cleanup
2018-12-03 18:24:07,426 INFO Connecting to TRex (127.0.0.1)...
2018-12-03 18:24:07,575 INFO Connected to TRex
2018-12-03 18:24:07,575 INFO    Port 0: Ethernet Controller XL710 for 40GbE QSFP+␣
↪speed=40Gbps mac=3c:fd:fe:b5:3d:70 pci=0000:5e:00.0 driver=net_i40e
2018-12-03 18:24:07,575 INFO    Port 1: Ethernet Controller XL710 for 40GbE QSFP+␣
↪speed=40Gbps mac=3c:fd:fe:b5:3d:71 pci=0000:5e:00.1 driver=net_i40e
2018-12-03 18:24:07,626 INFO Found built-in VM image file nfvbenchvm-0.6.qcow2
2018-12-03 18:24:09,072 INFO Created flavor 'nfvbench.medium'
2018-12-03 18:24:10,004 INFO Created network: nfvbench-lnet.
2018-12-03 18:24:10,837 INFO Created network: nfvbench-rnet.
2018-12-03 18:24:12,065 INFO Security disabled on port nfvbench-loop-vm0-0
2018-12-03 18:24:13,425 INFO Security disabled on port nfvbench-loop-vm0-1
2018-12-03 18:24:13,425 INFO Creating instance nfvbench-loop-vm0 with AZ
2018-12-03 18:24:16,052 INFO Created instance nfvbench-loop-vm0 - waiting for␣
↪placement resolution...
2018-12-03 18:24:16,240 INFO Waiting for instance nfvbench-loop-vm0 to become active␣
↪(retry 1/101)...
<snip>
2018-12-03 18:24:59,266 INFO Waiting for instance nfvbench-loop-vm0 to become active␣
↪(retry 21/101)...
2018-12-03 18:25:01,427 INFO Instance nfvbench-loop-vm0 is active and has been placed␣
↪on nova:charter-compute-5
2018-12-03 18:25:02,819 INFO Security disabled on port nfvbench-loop-vm1-0
2018-12-03 18:25:04,198 INFO Security disabled on port nfvbench-loop-vm1-1
2018-12-03 18:25:04,199 INFO Creating instance nfvbench-loop-vm1 with AZ nova:charter-
↪compute-5
2018-12-03 18:25:05,032 INFO Created instance nfvbench-loop-vm1 on nova:charter-
↪compute-5
2018-12-03 18:25:05,033 INFO Instance nfvbench-loop-vm0 is ACTIVE on nova:charter-
↪compute-5
```

(continues on next page)

(continued from previous page)

```
2018-12-03 18:25:05,212 INFO Waiting for 1/2 instance to become active (retry 1/100)..
↪.
<snip>
2018-12-03 18:25:48,531 INFO Waiting for 1/2 instance to become active (retry 21/100).
↪..
2018-12-03 18:25:50,677 INFO Instance nfvbench-loop-vm1 is ACTIVE on nova:charter-
↪compute-5
2018-12-03 18:25:50,677 INFO All instances are active
2018-12-03 18:25:50,677 INFO Port 0: VLANs [3830, 3830]
2018-12-03 18:25:50,677 INFO Port 1: VLANs [3831, 3831]
2018-12-03 18:25:50,677 INFO Port 0: dst MAC ['fa:16:3e:de:4e:54', 'fa:16:3e:7a:26:2b
↪']
2018-12-03 18:25:50,677 INFO Port 1: dst MAC ['fa:16:3e:6c:bb:cd', 'fa:16:3e:e0:48:45
↪']
2018-12-03 18:25:50,678 INFO ChainRunner initialized
2018-12-03 18:25:50,678 INFO Starting 2xPVP benchmark...
2018-12-03 18:25:50,683 INFO Starting traffic generator to ensure end-to-end␣
↪connectivity
2018-12-03 18:25:50,698 INFO Created 2 traffic streams for port 0.
2018-12-03 18:25:50,700 INFO Created 2 traffic streams for port 1.
2018-12-03 18:25:50,821 INFO Captured unique src mac 0/4, capturing return packets␣
↪(retry 1/100)...
2018-12-03 18:25:52,944 INFO Received packet from mac: fa:16:3e:de:4e:54 (chain=0,␣
↪port=0)
2018-12-03 18:25:52,945 INFO Received packet from mac: fa:16:3e:6c:bb:cd (chain=0,␣
↪port=1)
2018-12-03 18:25:53,077 INFO Captured unique src mac 2/4, capturing return packets␣
↪(retry 2/100)...
<snip>
2018-12-03 18:26:10,798 INFO End-to-end connectivity established
2018-12-03 18:26:10,816 INFO Cleared all existing streams
2018-12-03 18:26:10,846 INFO Created 4 traffic streams for port 0.
2018-12-03 18:26:10,849 INFO Created 4 traffic streams for port 1.
2018-12-03 18:26:10,849 INFO Starting to generate traffic...
2018-12-03 18:26:10,850 INFO Running traffic generator
2018-12-03 18:26:11,877 INFO TX: 10000004; RX: 9999999; Est. Dropped: 5; Est. Drop␣
↪rate: 0.0000%
2018-12-03 18:26:11,877 INFO ...traffic generating ended.
2018-12-03 18:26:11,882 INFO Service chain 'PVP' run completed.
2018-12-03 18:26:11,936 INFO Clean up skipped.
2018-12-03 18:26:11,969 INFO
========== NFVBench Summary ==========
Date: 2018-12-03 18:25:50
NFVBench version 3.0.3.dev1
Openstack Neutron:
  vSwitch: OPENVSWITCH
  Encapsulation: VLAN
Benchmarks:
> Networks:
  > Components:
    > Traffic Generator:
        Profile: trex-local
        Tool: TRex
    > Versions:
      > Traffic_Generator:
          build_date: Nov 13 2017
          version: v2.32
```

(continues on next page)

(continued from previous page)

```
          built_by: hhaim
          mode: STL
          build_time: 10:58:17
      > CiscoVIM: 2.9.7-17036
  > Service chain:
    > PVP:
      > Traffic:
          Profile: traffic_profile_64B
          Bidirectional: True
          Flow count: 10000
          Service chains count: 2
          Compute nodes: [u'nova:charter-compute-5']

          Run Summary:

            +-----------------+-------------+----------------------+------------------
↪----+----------------------+
            |   L2 Frame Size |  Drop Rate  |   Avg Latency (usec) |   Min Latency␣
↪(usec) |   Max Latency (usec) |
           ␣
↪+=================+=============+======================+======================+=================
            |              64 |   0.0000%   |                   13 |                  ␣
↪ 10 |                  141 |
            +-----------------+-------------+----------------------+------------------
↪----+----------------------+


          L2 frame size: 64

          Run Config:

            +-------------+---------------------------+------------------------+------
↪-----------+---------------------------+------------------------+-----------------+
            |  Direction  |  Requested TX Rate (bps)  |  Actual TX Rate (bps)  |  RX␣
↪Rate (bps)  |  Requested TX Rate (pps)  |  Actual TX Rate (pps)  |  RX Rate (pps)  |
           ␣
↪+=============+===========================+========================+==================+=============
            |   Forward   |       336.0000 Mbps       |     336.0000 Mbps      |  336.
↪0000 Mbps  |        500,000 pps        |      500,000 pps       |   500,000 pps   |
            +-------------+---------------------------+------------------------+------
↪-----------+---------------------------+------------------------+-----------------+
            |   Reverse   |       336.0000 Mbps       |     336.0000 Mbps      |  336.
↪0000 Mbps  |        500,000 pps        |      500,000 pps       |   500,000 pps   |
            +-------------+---------------------------+------------------------+------
↪-----------+---------------------------+------------------------+-----------------+
            |    Total    |       672.0000 Mbps       |     672.0000 Mbps      |  672.
↪0000 Mbps  |       1,000,000 pps       |     1,000,000 pps      |  1,000,000 pps  |
            +-------------+---------------------------+------------------------+------
↪-----------+---------------------------+------------------------+-----------------+

          Forward Chain Packet Counters and Latency:

            +---------+--------------+--------------+------------+------------+-------
↪------+
            |  Chain  |  TRex.TX.p0  |  TRex.RX.p1  |  Avg lat.  |  Min lat.  |  Max␣
↪lat.  |
           ␣
↪+=========+==============+==============+============+============+=======
```

(continues on next page)

(continued from previous page)

```
            |    0    |   250,000    |   250,000    |  17 usec   |  10 usec   |  138␣
↪usec  |
            +---------+--------------+--------------+------------+------------+-------
↪-----+
            |    1    |   250,000    |   250,000    |  17 usec   |  10 usec   |  139␣
↪usec  |
            +---------+--------------+--------------+------------+------------+-------
↪-----+
            |  total  |   500,000    |   500,000    |  17 usec   |  10 usec   |  139␣
↪usec  |
            +---------+--------------+--------------+------------+------------+-------
↪-----+

          Reverse Chain Packet Counters and Latency:

            +---------+--------------+--------------+------------+------------+-------
↪-----+
            |  Chain  |  TRex.TX.p1  |  TRex.RX.p0  |  Avg lat.  |  Min lat.  |  Max␣
↪lat.  |
          ␣
↪+=========+==============+==============+============+============+============+
            |    0    |   250,000    |   250,000    |  12 usec   |  10 usec   |  141␣
↪usec  |
            +---------+--------------+--------------+------------+------------+-------
↪-----+
            |    1    |   250,000    |   250,000    |  11 usec   |  10 usec   |  132␣
↪usec  |
            +---------+--------------+--------------+------------+------------+-------
↪-----+
            |  total  |   500,000    |   500,000    |  12 usec   |  10 usec   |  141␣
↪usec  |
            +---------+--------------+--------------+------------+------------+-------
↪-----+
```

Example of configuration file (non shared network)

Multi-chain with non shared network (requires 2 segmentation ID per chain), example with 2 chains sharing the same 2 SRIOV ports (or PF):

```
flavor:
   # Number of vCPUs for the flavor
   vcpus: 4
   # Memory for the flavor in MB
   ram: 8192
   # Size of local disk in GB
   disk: 0
   extra_specs:
      "hw:cpu_policy": dedicated
cores: 8
sriov: true
internal_networks:
   left:
        segmentation_id: [3830, 3831]
        physical_network: phys_sriov0
   right:
```

(continues on next page)

(continued from previous page)

```
        segmentation_id: [3832, 3833]
        physical_network: phys_sriov1
```

Example of full run 2xPVP non-shared network SR-IOV:

```
2018-12-04 17:15:25,284 INFO -c /tmp/nfvbench/sriov.yaml --rate 1Mpps --duration 1 -
↪scc 2 --no-cleanup
2018-12-04 17:15:25,287 INFO Connecting to TRex (127.0.0.1)...
2018-12-04 17:15:25,463 INFO Connected to TRex
2018-12-04 17:15:25,464 INFO    Port 0: Ethernet Controller XL710 for 40GbE QSFP+␣
↪speed=40Gbps mac=3c:fd:fe:b5:3d:70 pci=0000:5e:00.0 driver=net_i40e
2018-12-04 17:15:25,464 INFO    Port 1: Ethernet Controller XL710 for 40GbE QSFP+␣
↪speed=40Gbps mac=3c:fd:fe:b5:3d:71 pci=0000:5e:00.1 driver=net_i40e
2018-12-04 17:15:25,515 INFO Found built-in VM image file nfvbenchvm-0.6.qcow2
2018-12-04 17:15:26,457 INFO Created flavor 'nfvbench.medium'
2018-12-04 17:15:27,449 INFO Created network: nfvbench-lnet0.
2018-12-04 17:15:28,368 INFO Created network: nfvbench-rnet0.
2018-12-04 17:15:29,143 INFO Created port nfvbench-loop-vm0-0
2018-12-04 17:15:29,626 INFO Security disabled on port nfvbench-loop-vm0-0
2018-12-04 17:15:30,636 INFO Created port nfvbench-loop-vm0-1
2018-12-04 17:15:31,139 INFO Security disabled on port nfvbench-loop-vm0-1
2018-12-04 17:15:31,140 INFO Creating instance nfvbench-loop-vm0 with AZ
2018-12-04 17:15:34,893 INFO Created instance nfvbench-loop-vm0 - waiting for␣
↪placement resolution...
2018-12-04 17:15:35,068 INFO Waiting for instance nfvbench-loop-vm0 to become active␣
↪(retry 1/101)...
<snip>
2018-12-04 17:16:22,253 INFO Instance nfvbench-loop-vm0 is active and has been placed␣
↪on nova:charter-compute-4
2018-12-04 17:16:23,154 INFO Created network: nfvbench-lnet1.
2018-12-04 17:16:23,863 INFO Created network: nfvbench-rnet1.
2018-12-04 17:16:24,799 INFO Created port nfvbench-loop-vm1-0
2018-12-04 17:16:25,267 INFO Security disabled on port nfvbench-loop-vm1-0
2018-12-04 17:16:26,006 INFO Created port nfvbench-loop-vm1-1
2018-12-04 17:16:26,612 INFO Security disabled on port nfvbench-loop-vm1-1
2018-12-04 17:16:26,612 INFO Creating instance nfvbench-loop-vm1 with AZ nova:charter-
↪compute-4
2018-12-04 17:16:27,610 INFO Created instance nfvbench-loop-vm1 on nova:charter-
↪compute-4
2018-12-04 17:16:27,610 INFO Instance nfvbench-loop-vm0 is ACTIVE on nova:charter-
↪compute-4
2018-12-04 17:16:27,788 INFO Waiting for 1/2 instance to become active (retry 1/100)..
↪.
<snip>

2018-12-04 17:17:04,258 INFO Instance nfvbench-loop-vm1 is ACTIVE on nova:charter-
↪compute-4
2018-12-04 17:17:04,258 INFO All instances are active
2018-12-04 17:17:04,259 INFO Port 0: VLANs [3830, 3831]
2018-12-04 17:17:04,259 INFO Port 1: VLANs [3832, 3833]
2018-12-04 17:17:04,259 INFO Port 0: dst MAC ['fa:16:3e:ef:f4:b0', 'fa:16:3e:e5:74:cd
↪']
2018-12-04 17:17:04,259 INFO Port 1: dst MAC ['fa:16:3e:d6:dc:84', 'fa:16:3e:8e:d9:30
↪']
2018-12-04 17:17:04,259 INFO ChainRunner initialized
2018-12-04 17:17:04,260 INFO Starting 2xPVP benchmark...
2018-12-04 17:17:04,266 INFO Starting traffic generator to ensure end-to-end␣
↪connectivity
```

(continues on next page)

(continued from previous page)

```
2018-12-04 17:17:04,297 INFO Created 2 traffic streams for port 0.
2018-12-04 17:17:04,300 INFO Created 2 traffic streams for port 1.
2018-12-04 17:17:04,420 INFO Captured unique src mac 0/4, capturing return packets␣
↪(retry 1/100)...
2018-12-04 17:17:06,532 INFO Received packet from mac: fa:16:3e:d6:dc:84 (chain=0,␣
↪port=1)
2018-12-04 17:17:06,532 INFO Received packet from mac: fa:16:3e:ef:f4:b0 (chain=0,␣
↪port=0)
2018-12-04 17:17:06,644 INFO Captured unique src mac 2/4, capturing return packets␣
↪(retry 2/100)...
<snip>

2018-12-04 17:17:24,337 INFO Received packet from mac: fa:16:3e:8e:d9:30 (chain=1,␣
↪port=1)
2018-12-04 17:17:24,338 INFO Received packet from mac: fa:16:3e:e5:74:cd (chain=1,␣
↪port=0)
2018-12-04 17:17:24,338 INFO End-to-end connectivity established
2018-12-04 17:17:24,355 INFO Cleared all existing streams
2018-12-04 17:17:24,383 INFO Created 4 traffic streams for port 0.
2018-12-04 17:17:24,386 INFO Created 4 traffic streams for port 1.
2018-12-04 17:17:24,386 INFO Starting to generate traffic...
2018-12-04 17:17:24,386 INFO Running traffic generator
2018-12-04 17:17:25,415 INFO TX: 1000004; RX: 1000004; Est. Dropped: 0; Est. Drop␣
↪rate: 0.0000%
2018-12-04 17:17:25,415 INFO ...traffic generating ended.
2018-12-04 17:17:25,420 INFO Service chain 'PVP' run completed.
2018-12-04 17:17:25,471 INFO Clean up skipped.
2018-12-04 17:17:25,508 INFO
========== NFVBench Summary ==========
Date: 2018-12-04 17:17:04
NFVBench version 3.0.3.dev1
Openstack Neutron:
  vSwitch: OPENVSWITCH
  Encapsulation: VLAN
Benchmarks:
> Networks:
  > Components:
    > Traffic Generator:
        Profile: trex-local
        Tool: TRex
    > Versions:
      > Traffic_Generator:
          build_date: Nov 13 2017
          version: v2.32
          built_by: hhaim
          mode: STL
          build_time: 10:58:17
      > CiscoVIM: 2.9.7-17036
  > Service chain:
    > PVP:
      > Traffic:
          Profile: traffic_profile_64B
          Bidirectional: True
          Flow count: 10000
          Service chains count: 2
          Compute nodes: [u'nova:charter-compute-4']
```

(continues on next page)

(continued from previous page)

```
            Run Summary:

              +-----------------+-------------+----------------------+----------------
↪------+----------------------+
              |   L2 Frame Size |  Drop Rate  |   Avg Latency (usec) |   Min Latency␣
↪(usec) |   Max Latency (usec) |
              ␣
↪+=================+=============+======================+======================+=================
              |              64 |   0.0000%   |                   18 |              ␣
↪   10 |                  120 |
              +-----------------+-------------+----------------------+----------------
↪------+----------------------+


            L2 frame size: 64

            Run Config:

              +-------------+---------------------------+-----------------------+----
↪-------------+---------------------------+-----------------------+-----------------
↪+
              |  Direction  |  Requested TX Rate (bps)  |  Actual TX Rate (bps)  | ␣
↪RX Rate (bps)  |  Requested TX Rate (pps)  |  Actual TX Rate (pps)  |  RX Rate␣
↪(pps)  |
              ␣
↪+=============+===========================+=======================+=================+=============
              |   Forward   |       336.0000 Mbps       |     336.0013 Mbps      | ␣
↪336.0013 Mbps  |        500,000 pps        |      500,002 pps       |   500,002 pps␣
↪  |
              +-------------+---------------------------+-----------------------+----
↪-------------+---------------------------+-----------------------+-----------------
↪+
              |   Reverse   |       336.0000 Mbps       |     336.0013 Mbps      | ␣
↪336.0013 Mbps  |        500,000 pps        |      500,002 pps       |   500,002 pps␣
↪  |
              +-------------+---------------------------+-----------------------+----
↪-------------+---------------------------+-----------------------+-----------------
↪+
              |    Total    |       672.0000 Mbps       |     672.0027 Mbps      | ␣
↪672.0027 Mbps  |       1,000,000 pps       |     1,000,004 pps      |  1,000,004␣
↪pps  |
              +-------------+---------------------------+-----------------------+----
↪-------------+---------------------------+-----------------------+-----------------
↪+

            Forward Chain Packet Counters and Latency:

              +---------+--------------+--------------+------------+------------+-----
↪-------+
              |  Chain  |  TRex.TX.p0  |  TRex.RX.p1  |  Avg lat.  |  Min lat.  | ␣
↪Max lat.  |
              ␣
↪+=========+==============+==============+============+============+=============+
              |    0    |   250,001    |   250,001    |  26 usec   |  10 usec   |  70␣
↪usec   |
              +---------+--------------+--------------+------------+------------+-----
↪-------+
```

(continues on next page)

(continued from previous page)

```
              |    1    |   250,001    |   250,001    |  11 usec   |  10 usec   |  39 usec   |
              +---------+--------------+--------------+------------+------------+------------+
              |  total  |   500,002    |   500,002    |  19 usec   |  10 usec   |  70 usec   |
              +---------+--------------+--------------+------------+------------+------------+

            Reverse Chain Packet Counters and Latency:

              +---------+--------------+--------------+------------+------------+------------+
              |  Chain  |  TRex.TX.p1  |  TRex.RX.p0  |  Avg lat.  |  Min lat.  |  Max lat.  |
              +=========+==============+==============+============+============+============+
              |    0    |   250,001    |   250,001    |  19 usec   |  10 usec   | 119 usec   |
              +---------+--------------+--------------+------------+------------+------------+
              |    1    |   250,001    |   250,001    |  19 usec   |  10 usec   | 120 usec   |
              +---------+--------------+--------------+------------+------------+------------+
              |  total  |   500,002    |   500,002    |  19 usec   |  10 usec   | 120 usec   |
              +---------+--------------+--------------+------------+------------+------------+
```

7.1.15 NFVbench Server mode and NFVbench client API

HTTP Interface

<http-url>/echo (GET)

This request simply returns whatever content is sent in the body of the request (body should be in json format, only used for testing)

Example request:

```
curl -XGET '127.0.0.1:7555/echo' -H "Content-Type: application/json" -d '{"nfvbench":
 ↪"test"}'
Response:
{
  "nfvbench": "test"
}
```

<http-url>/status (GET)

This request fetches the status of an asynchronous run. It will return in json format:

- a request pending reply (if the run is still not completed)
- an error reply if there is no run pending
- or the complete result of the run

The client can keep polling until the run completes.

Example of return when the run is still pending:

```
{
  "error_message": "nfvbench run still pending",
  "status": "PENDING"
}
```

Example of return when the run completes:

```
{
  "result": {...}
  "status": "OK"
}
```

<http-url>/start_run (POST)

This request starts an NFVBench run with passed configurations. If no configuration is passed, a run with default configurations will be executed.

Example request: curl -XPOST 'localhost:7556/start_run' -H "Content-Type: application/json" -d @nfvbenchconfig.json

See "NFVbench configuration JSON parameter" below for details on how to format this parameter.

The request returns immediately with a json content indicating if there was an error (status=ERROR) or if the request was submitted successfully (status=PENDING). Example of return when the submission is successful:

```
{
  "error_message": "NFVbench run still pending",
  "request_id": "42cccb7effdc43caa47f722f0ca8ec96",
  "status": "PENDING"
}
```

If there is already an NFVBench running then it will return:

```
{
 "error_message": "there is already an NFVbench request running",
 "status": "ERROR"
}
```

NFVbench configuration JSON parameter

The NFVbench configuration describes the parameters of an NFVbench run and can be passed to the NFVbench server as a JSON document.

Default configuration

The simplest JSON document is the empty dictionary "{}" which indicates to use the default NFVbench configuration:

- PVP
- NDR-PDR measurement
- 64 byte packets
- 1 flow per direction

The entire default configuration can be viewed using the –show-json-config option on the cli:

```
# nfvbench --show-config
{
    "availability_zone": null,
    "compute_node_user": "root",
    "compute_nodes": null,
    "debug": false,
    "duration_sec": 60,
    "flavor": {
        "disk": 0,
        "extra_specs": {
            "hw:cpu_policy": "dedicated",
            "hw:mem_page_size": 2048
        },
        "ram": 8192,
        "vcpus": 2
    },
    "flavor_type": "nfv.medium",
    "flow_count": 1,
    "generic_poll_sec": 2,
    "generic_retry_count": 100,
    "inter_node": false,
    "internal_networks": {
        "left": {
            "name": "nfvbench-net0",
            "subnet": "nfvbench-subnet0",
            "cidr": "192.168.1.0/24",
        },
        "right": {
            "name": "nfvbench-net1",
            "subnet": "nfvbench-subnet1",
            "cidr": "192.168.2.0/24",
        },
        "middle": {
            "name": "nfvbench-net2",
            "subnet": "nfvbench-subnet2",
            "cidr": "192.168.3.0/24",
        }
    },
    "interval_sec": 10,
```

(continues on next page)

(continued from previous page)


```
    "json": null,
    "loop_vm_name": "nfvbench-loop-vm",
    "measurement": {
        "NDR": 0.001,
        "PDR": 0.1,
        "load_epsilon": 0.1
    },
    "name": "(built-in default config)",
    "no_cleanup": false,
    "no_traffic": false,
    "openrc_file": "/tmp/nfvbench/openstack/openrc",
    "rate": "ndr_pdr",
    "service_chain": "PVP",
    "service_chain_count": 1,
    "sriov": false,
    "std_json": null,
    "traffic": {
        "bidirectional": true,
        "profile": "traffic_profile_64B"
    },
    "traffic_generator": {
        "default_profile": "trex-local",
        "gateway_ip_addrs": [
            "1.1.0.2",
            "2.2.0.2"
        ],
        "gateway_ip_addrs_step": "0.0.0.1",
        "generator_profile": [
            {
                "cores": 3,
                "interfaces": [
                    {
                        "pci": "0a:00.0",
                        "port": 0,
                        "switch_port": "Ethernet1/33",
                        "vlan": null
                    },
                    {
                        "pci": "0a:00.1",
                        "port": 1,
                        "switch_port": "Ethernet1/34",
                        "vlan": null
                    }
                ],
                "intf_speed": null,
                "ip": "127.0.0.1",
                "name": "trex-local",
                "tool": "TRex"
            }
        ],
        "host_name": "nfvbench_tg",
        "ip_addrs": [
            "10.0.0.0/8",
            "20.0.0.0/8"
        ],
        "ip_addrs_step": "0.0.0.1",
        "mac_addrs": [
```


(continues on next page)

(continued from previous page)

```
            "00:10:94:00:0A:00",
            "00:11:94:00:0A:00"
        ],
        "step_mac": null,
        "tg_gateway_ip_addrs": [
            "1.1.0.100",
            "2.2.0.100"
        ],
        "tg_gateway_ip_addrs_step": "0.0.0.1"
    },
    "traffic_profile": [
        {
            "l2frame_size": [
                "64"
            ],
            "name": "traffic_profile_64B"
        },
        {
            "l2frame_size": [
                "IMIX"
            ],
            "name": "traffic_profile_IMIX"
        },
        {
            "l2frame_size": [
                "1518"
            ],
            "name": "traffic_profile_1518B"
        },
        {
            "l2frame_size": [
                "64",
                "IMIX",
                "1518"
            ],
            "name": "traffic_profile_3sizes"
        }
    ],
    "unidir_reverse_traffic_pps": 1,
    "vlan_tagging": true,
}
```

Common examples of JSON configuration

Use the default configuration but use 10000 flows per direction (instead of 1):

```
{ "flow_count": 10000 }
```

Use default confguration but with 10000 flows, “EXT” chain and IMIX packet size:

```
{
   "flow_count": 10000,
   "service_chain": "EXT",
    "traffic": {
        "profile": "traffic_profile_IMIX"
```

(continues on next page)

(continued from previous page)

```
    },
}
```

A short run of 5 seconds at a fixed rate of 1Mpps (and everything else same as the default configuration):

```
{
   "duration": 5,
   "rate": "1Mpps"
}
```

Use the default configuration but force TRex restart:

```
{
   "restart": true
}
```

Example of interaction with the NFVbench server using HTTP and curl

HTTP requests can be sent directly to the NFVbench server from CLI using curl from any host that can connect to the server (here we run it from the local host).

This is a POST request to start a run using the default NFVbench configuration but with traffic generation disabled ("no_traffic" property is set to true):

```
[root@sjc04-pod3-mgmt ~]# curl -H "Accept: application/json" -H "Content-type:␣
↪application/json" -X POST -d '{"no_traffic":true}' http://127.0.0.1:7555/start_run
{
  "error_message": "nfvbench run still pending",
  "status": "PENDING"
}
[root@sjc04-pod3-mgmt ~]#
```

This request will return immediately with status set to "PENDING" if the request was started successfully.

The status can be polled until the run completes. Here the poll returns a "PENDING" status, indicating the run is still not completed:

```
[root@sjc04-pod3-mgmt ~]# curl -G http://127.0.0.1:7555/status
{
  "error_message": "nfvbench run still pending",
  "status": "PENDING"
}
[root@sjc04-pod3-mgmt ~]#
```

Finally, the status request returns a "OK" status along with the full results (truncated here):

```
[root@sjc04-pod3-mgmt ~]# curl -G http://127.0.0.1:7555/status
{
  "result": {
    "benchmarks": {
      "network": {
        "service_chain": {
          "PVP": {
            "result": {
              "bidirectional": true,
```

(continues on next page)

(continued from previous page)

```
              "compute_nodes": {
                "nova:sjc04-pod3-compute-4": {
                  "bios_settings": {
                    "Adjacent Cache Line Prefetcher": "Disabled",
                    "All Onboard LOM Ports": "Enabled",
                    "All PCIe Slots OptionROM": "Enabled",
                    "Altitude": "300 M",
...

    "date": "2017-03-31 22:15:41",
    "nfvbench_version": "0.3.5",
    "openstack_spec": {
      "encaps": "VxLAN",
      "vswitch": "VTS"
    }
  },
  "status": "OK"
}
[root@sjc04-pod3-mgmt ~]#
```

Example of interaction with the NFVbench server using a python CLI app (nfvbench_client)

The module client/client.py contains an example of python class that can be used to control the NFVbench server from a python app using HTTP.

The module client/nfvbench_client.py has a simple main application to control the NFVbench server from CLI. The "nfvbench_client" wrapper script can be used to invoke the client front end (this wrapper is pre-installed in the NFVbench container)

Example of invocation of the nfvbench_client front end, from the host (assume the name of the NFVbench container is "nfvbench"), use the default NFVbench configuration but do not generate traffic (no_traffic property set to true, the full json result is truncated here):

```
[root@sjc04-pod3-mgmt ~]# docker exec -it nfvbench nfvbench_client -c '{"no_traffic
↪":true}' http://127.0.0.1:7555
{u'status': u'PENDING', u'error_message': u'nfvbench run still pending'}
{u'status': u'PENDING', u'error_message': u'nfvbench run still pending'}
{u'status': u'PENDING', u'error_message': u'nfvbench run still pending'}

{u'status': u'OK', u'result': {u'date': u'2017-03-31 22:04:59', u'nfvbench_version': u
↪'0.3.5',
 u'config': {u'compute_nodes': None, u'compute_node_user': u'root', u'traffic_
↪generator': {u'tg_gateway_ip_addrs': [u'1.1.0.100', u'2.2.0.100'], u'ip_addrs_step
↪': u'0.0.0.1',
 u'step_mac': None, u'generator_profile': [{u'intf_speed': u'', u'interfaces': [{u'pci
↪': u'0a:00.0', u'port': 0, u'vlan': 1998, u'switch_port': None},

...

[root@sjc04-pod3-mgmt ~]#
```

7.1.16 Frequently Asked Questions

General Questions

Can NFVbench be used without OpenStack

Yes. This can be done using the EXT chain mode, with or without ARP (depending on whether your systen under test can do routing) and by setting the openrc_file property to empty in the NFVbench configuration.

Can NFVbench be used with a different traffic generator than TRex?

This is possible but requires developing a new python class to manage the new traffic generator interface.

Can I connect Trex directly to my compute node?

Yes.

Can I drive NFVbench using a REST interface?

NFVbench can run in server mode and accept HTTP requests to run any type of measurement (fixed rate run or NDR_PDR run) with any run configuration.

Can I run NFVbench on a Cisco UCS-B series blade?

Yes provided your UCS-B series server has a Cisco VIC 1340 (with a recent firmware version). TRex will require VIC firmware version 3.1(2) or higher for blade servers (which supports more filtering capabilities). In this setting, the 2 physical interfaces for data plane traffic are simply hooked to the UCS-B fabric interconnect (no need to connect to a switch).

Troubleshooting

TrafficClientException: End-to-end connectivity cannot be ensured

Prior to running a benchmark, NFVbench will make sure that traffic is passing in the service chain by sending a small flow of packets in each direction and verifying that they are received back at the other end. This exception means that NFVbench cannot pass any traffic in the service chain.

The most common issues that prevent traffic from passing are: - incorrect wiring of the NFVbench/TRex interfaces - incorrect vlan_tagging setting in the NFVbench configuration, this needs to match how the NFVbench ports on the switch are configured (trunk or access port)

- if the switch port is configured as access port, you must disable vlan_tagging in the NFVbench configuration
- if the switch port is configured as trunk (recommended method), you must enable it

Issues with high performances at a high line rate

Flow statistics and/or latency stream can cause performance issue when testing high line rate.

Flow statistics implies CPU usage to analyse packets and retrieve statistics. CPU can reach 100% usage when high throughput is tested because only one CPU is used for packet reception in TRex. The `--no-flow-stats` option allows you to disable TRex statistics aggregation during the NFVBench test. This, will permit to save CPU capabilities used for packet reception.

Example of use :

```
nfvbench ``--no-flow-stats``

2019-10-28 10:26:52,099 INFO End-to-end connectivity established
2019-10-28 10:26:52,127 INFO Cleared all existing streams
2019-10-28 10:26:52,129 INFO Traffic flow statistics are disabled.
```

Latency streams implies also CPU usage to analyse packets and retrieve latency values. CPU can reach 100% usage when high throughput is tested because only one CPU is used for packet reception in TRex. The `--no-latency-streams` option allows you to disable latency streams during the NFVBench test. This, will permit to save CPU capabilities used for packet reception but no latency information will be return (to be used only if latency value has no meaning for your test).

Example of use :

```
nfvbench ``--no-latency-streams``
2019-10-28 10:30:03,955 INFO End-to-end connectivity established
2019-10-28 10:30:03,982 INFO Cleared all existing streams
2019-10-28 10:30:03,982 INFO Latency streams are disabled
```

Latency flow statistics implies CPU usage to analyse packets and retrieve statistics. CPU can reach 100% usage when high throughput is tested because only one CPU is used for packet reception in TRex. The `--no-latency-stats` option allows you to disable TRex statistics aggregation for latency packets during the NFVBench test. This, will permit to save CPU capabilities used for packet reception.

Example of use :

```
nfvbench ``--no-latency-stats``
2019-10-28 10:28:21,559 INFO Cleared all existing streams
2019-10-28 10:28:21,567 INFO Latency flow statistics are disabled.
```