USER GUIDE

Intel Corporation

Packet pROcessing eXecution Engine (PROX) - Performance Characterization for NFVI User Guide

Authors

Yury Kylulin

Luc Provoost

Petar Torre

1 Introduction

Properly designed Network Functions Virtualization Infrastructure (NFVI) environments deliver high packet processing rates, which are also a required dependency for onboarded network functions. NFVI testing methodology typically includes both functionality testing and performance characterization testing, to ensure that NFVI both exposes the correct APIs and is capable of packet processing.

This document describes how to use open source software tools to automate peak traffic (also called saturation) throughput testing to characterize the performance of a containerized NFVI system.

The text and examples in this document are intended for architects and testing engineers for Communications Service Providers (CSPs) and their vendors. This document describes tools used during development to evaluate whether or not a containerized NFVI can perform the required rates of packet processing within set packet drop rates and latency percentiles.

This document is part of the Network Transformation Experience Kit, which is available at https://networkbuilders.intel.com/network-technologies/network-transformation-exp-kits.

Table of Contents

Figures

Tables

2 Overview

Traditionally, establishing repeatable, reliable performance characterization required highly-trained engineers in specially-prepared labs, usually equipped with proprietary hardware load generators and test suites. Such traditional characterization methodologies faced challenges measuring the virtualized environment while also taking into account the network workload needs (which are onboarded at a later stage). This document describes a method to automate a typically manual process which can require extensive and expensive test gear. Note that these tests are not a substitute for acceptance, validation, or compliance tests.

This document describes how to use open source tools to characterize the performance of a containerized NFVI system. The key software components are an engine called Packet pROcessing eXecution (PROX) and automation scripts.

The PROX engine is a Data Plane Development Kit* (DPDK*) application that performs packet operations in a highly configurable manner and displays performance statistics. In the setup described in this guide, the PROX engine is packaged as a bare metal container, however, in other environments it can be packaged as a container in Virtual Machines (VMs) or without containers in a GuestOS VM.

The automation test scripts cover options from simple test runs to more complex test cases.

2.1 Test Framework

The diagram below describes the design and operation of the automated test framework software. On the desired server nodes, we instantiate two PROX engines.

The control script performs the following tasks:

- Configures one engine as the Generator and the other as Swap.
- Starts the test at the target maximum throughput level.
- Uses binary search to find a peak traffic (saturation) throughput rate that is within acceptable latency and packet loss rates.
- Presents the final results on screen.
- Sends the results as metrics to Prometheus* Pushgateway.
- Prometheus reads the metrics and presents them in Grafana*.

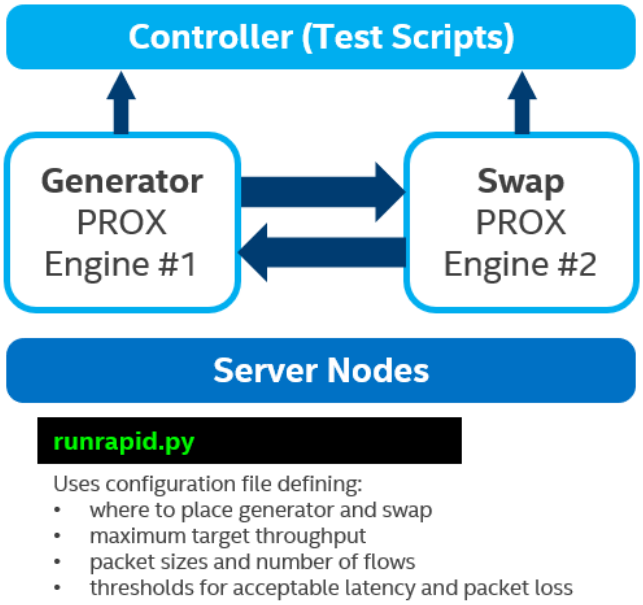


Figure 1. Test Framework Software

Note: The testing methodology and tools described here are for development purposes. They are not recommended for tests such as formal validation or acceptance test, analysis of different types of Quality of Service (QoS) traffic, measurement of packet delay variations (peak and average), individual subscriber QoS, burst propagation, shaping/scheduling, or 3GPP compliance tests.

2.2 Terminology

Table 1. Terminology

ABBREVIATION	DESCRIPTION
CNF	Cloud native Network Function
CSP	Communications Service Provider
DPDK*	Data Plane Development Kit*
K8s*	Kubernetes*
NFV	Network Functions Virtualization
NFVI	Network Functions Virtualization Infrastructure
OPNFV*	Open Platform for NFV*
PROX	Packet pROcessing eXecution engine
SR-IOV	Single Root Input/Output Virtualization
QoS	Quality of Service
VM	Virtual Machine

2.3 Reference Documentation

Table 2. Reference Documents

REFERENCE	SOURCE
Readme with requirements	https://git.opnfv.org/samplevnf/plain/VNFs/DPPD-PROX/helper-scripts/rapid/README.k8s
Packet pROcessing eXecution engine (PROX) documentation	https://wiki.opnfv.org/pages/viewpage.action?pageId=12387840
Automation script code	https://git.opnfv.org/samplevnf/tree/VNFs/DPPD-PROX/helper-scripts/rapid
Automation script documentation	https://wiki.opnfv.org/display/SAM/Rapid+scripting
Container Bare Metal for 2nd Generation Intel® Xeon® Scalable Processor Reference Architecture	https://builders.intel.com/docs/networkbuilders/container-bare-metal-for-2nd-generation-intel-xeon-scalable-processor.pdf
Container Bare Metal for 2nd Generation Intel® Xeon® Scalable Processor Reference Architecture Setup Scripts	https://github.com/intel/container-experience-kits

3 Ingredients

This section lists the hardware and software that were used to develop this document. You can use the same setup or a similar one, depending on your needs.

Note: These tools are designed for the development stage. Current versions have not completed rigorous security and other validation tests required for software used in production environments.

3.1 Hardware Bill of Materials

The hardware setup includes the following:

- Servers with 2x Intel® Xeon® Gold 6248 Processors and 2x Intel® Ethernet Network Adapter XXV710
- Servers are connected via Ethernet switch with 25 Gb ports

3.2 Software Bill of Materials

The software environment and tools include the following:

- OPNFV* Sample VNF PROX (packaged as CNF), version as of March 2020
- Single-root input/output virtualization (SR-IOV) Device Plugins for Kubernetes*, version as of March 2020
- Kubernetes v1.13 or later, for example v1.17.3
- Docker* v18.06.2 or later, for example v19.03.2
- CentOS* 7 x86_64 19.08 or Ubuntu* v19.10
- Prometheus* Push Gateway v1.1.0, Prometheus v2.16.0, and Grafana v6.6.2, or later.

Perform the steps in the next section to prepare the tools.

4 Setup

This section explains how to prepare the PROX tool for performance characterization of NFVI when running bare metal containers.

If you want to measure similar performance characterization tests of NFVI where containers run in VMs, then you must correctly map the data path up to the point where packets are delivered inside the VMs, and from there into pods where the containerized PROX runs.

4.1 Kubernetes Prerequisites for Sensitive Workload Placement

Prerequisites for environments where the testing tools will be built:

- Linux* OS supported by DPDK* (in this document we used CentOS* 7 and tried Ubuntu* v19.10)
- Python* v2.7, PIP to install Python modules as per .py files with import lines
- Working cluster with Kubernetes. Set up the cluser using Intel Container Bare Metal Reference Architecture version 1.17 or later (https://github.com/intel/container-experience-kits) or commercial distributions supporting the required features, such as Red Hat* OpenShift* v4.3.

Prerequisites for nodes running the tests:

- 1024x 2M Huge Pages configured on the nodes.
- SR-IOV Network Device Plugin for Kubernetes installed from: https://github.com/intel/sriov-network-device-plugin
- SR-IOV VFs configured. If "`0000\:18\:00.0`" is the PF device, this can be done with the command:
  ```
  echo 16 > /sys/bus/pci/devices/0000\:18\:00.0/sriov_numvfs
  ```
- Rebind SR-IOV to the vfio-pci module.
  As an example, SR-IOV VFs (rebind to the vfio-pci driver) pool is named: `intel.com/intel_sriov_vfio`.
  The network attachment definition is named: `k8s.v1.cni.cncf.io/networks: intel-sriov-vfio`
- If layer 2 tests are planned to be executed, then MAC addresses must be preconfigured for the SR-IOV VFs to avoid issues with randomly generated MAC addresses each time when the PROX starts. If "`enp24s0f0`" is the PF device, this can be done with:
  ```
  for i in {0..15}; do ih=$( printf "%x" $i ); ip link set dev enp24s0f0 vf $i mac
  02:00:00:22:33:0$ih; done
  ```

4.2 Build PROX Containerized Image

The following steps were performed on CentOS 7 x86_64 19.08.

On the server or VM where the test tool images should be prepared, use the commands:

```
sudo bash
git clone https://github.com/opnfv/samplevnf.git
cd samplevnf/VNFs/DPPD-PROX/helper-scripts/rapid
./dockerimage.sh build
```

The output should end with these lines:

```
Successfully tagged prox_slim:latest
Saving image prox_slim:latest to ./prox_slim.tar
```

You can see the image size with the command:

```
docker image ls prox_slim
```

The output shows:

```
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
prox_slim           latest              14299ef5b6ce        2 minutes ago       278MB
```

which is the size of the base CentOS image plus DPDK and PROX binaries, and scripts. Note that exact size varies based on your host OS.

4.3 Push Image to Repository

On one of the nodes, copy `prox_slim.tar` and load it into the node's local repository with the command:

```
./dockerimage.sh push
curl http://localhost:5000/v2/_catalog | grep prox_slim
```

```
{"repositories":["prox_slim"]}
```

If you use another image repository, then appropriately tag and push the image into it. For example, if you use Docker* Hub*, use the command: `docker push username/reponame` for the compressed size of ~82MB.

4.4 Configure Metrics Collection and Visualization

In this process, Docker configures metrics collection and visualization, the test automation script sends metrics to Pushgateway, Prometheus* collects the metrics, and then Grafana presents it. If you are using a different environment, then you can set up a similar process after Pushgateway. If you use another system to collect metrics, then modify `runrapid.py` for your environment.

The steps below assume the following:

- The user has privileges to run `docker` commands.
- Commands are run on the node that will run Pushgateway, with example IP address: `PUSHGWIP`
- The working directory is: `$WORK_DIR`

1. Perform the following one-time setup procedure:

```
cd $WORK_DIR
mkdir scripts && cd scripts
cat > run_once << EOF
#!/bin/bash -v
docker run --name grafana -d --network host -e "GF_SECURITY_ADMIN_PASSWORD=password"
grafana/grafana
EOF
cat > start_all << EOF
#!/bin/bash -v
docker run --name pushgateway -d --network host prom/pushgateway
docker run --name prometheus -d --network host -v
$WORK_DIR/scripts/prometheus.yml:/etc/prometheus/prometheus.yml prom/prometheus
docker start grafana
EOF
cat > stop_all << EOF
#!/bin/bash -v
docker stop grafana
docker kill prometheus pushgateway
docker rm prometheus pushgateway
EOF
cat > prometheus.yml << EOF
global:
  scrape_interval: 2s
  evaluation_interval: 15s
scrape_configs:
  - job_name: 'pushgateway'
    static_configs:
    - targets: ['$PUSHGWIP:9091']
EOF
chmod 755 run_once start_all stop_all
```

2. Ensure that the file is properly formatted including spaces as per listing above, using the command:

```
cat prometheus.yml
```

3. Complete the one-time setup with the script:

```
./run_once
```

4. After the one-time setup is completed, use the following script for later test runs:

```
./start_all
```

5. Verify that Grafana, Prometheus, and Pushgateway are running with the command:

```
docker ps | grep -e grafana -e prometheus -e pushgateway
```

6. Using a browser, go to the Grafana page `http://$PUSHGWIP:3000` , log in with admin/password, and change the password.
7. Add the Prometheus data source with URL `http://$PUSHGWIP:9090` where:
   a. Replace `$PUSHGWIP` with your IP address.
   b. Enter the desired Scrape interval, for example 15s.
   c. Press the Save & Test button which should result in the message "*Data source is working*".

After these steps are complete, Pushgateway is ready to receive metrics from PROX tests. From there, Prometheus will collect them and Grafana will visualize them as graphs.

If you want to protect Pushgateway, then modify the Python script `runrapid.py` accordingly.

Later as needed, you can do cleanup on old metrics with the command:

```
./stop_all && ./start_all
```

The cleanup step above will not delete the Grafana configuration.

5 Prepare Pods and Run Tests

This section describes how to prepare test cases and run NFVI performance characterization. It assumes that the PROX image has been created and pushed to the image registry, and the metrics collection and visualization are ready.

5.1 Prepare PROX Pods

View available nodes with the command:

```
kubectl get nodes -o wide
```

From this output, choose the nodes where you want to run PROX pods and update the `rapid.pods` file with the correct IP addresses used for dataplane traffic (not those used for node management) to be similar to the following:

```
[DEFAULT]
total_number_of_pods=2

[POD1]
nodeSelector_hostname=node1
dp_ip=192.168.30.11

[POD2]
nodeSelector_hostname=node2
dp_ip=192.168.30.12
```

This will run the tests on two or more specified nodes. If you want to run the tests on the same node, then ensure that the Generator and Swap pods run on different cores. This can be configured in `*.test` depending on the chosen test case.

Create test pods with the command:

```
./createrapidk8s.py
```

Check that this succeeded with the command:

```
kubectl get pod | awk ' NR==1 { print $0 } $0~"pod-rapid" { print $0 } '
```

View the output and note that the STATUS should change to “Running”:

```
NAME          READY   STATUS    RESTARTS   AGE
pod-rapid-1   1/1     Running   0          53s
pod-rapid-2   1/1     Running   0          53s
```

5.2 Run Tests

To get results into Pushgateway, in the `basicrapid.test` file, uncomment the line with `#PushGateway` and enter the correct URL as shown in the example below:

```
PushGateway=http://192.168.0.235:9091
```

Start tests with the command:

```
./runrapid.py
```

This controller script will connect to the running PROX pods, run tests defined in the `basicrapid.test` file (or another `.test` file if it is specified), report results in the text screen, and send them into Pushgateway.

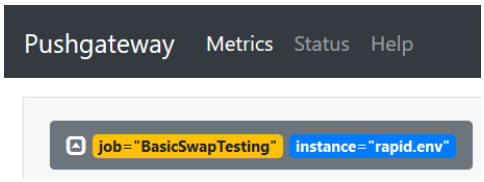
The following example shows that using the default test setup (including how many cores are used for PROX functionality), we observed ~13.3Mpps (of 64B size) sustained without exceeding the defined packet drop rate or latency percentile.

Note: If `runrapid.py` doesn’t start the PROX engines with the message “`Creating mempool…`” in your environment, then try increasing the huge pages memory allocation from 512Mi to 1Gi in `pod-rapid.yaml`.

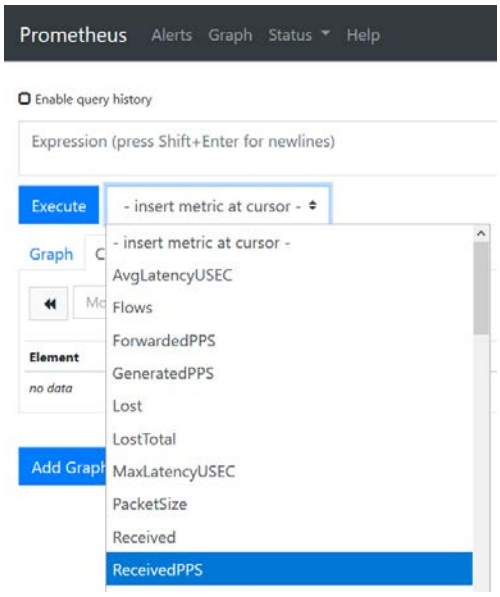
```
Using 'rapid.env' as name for the environment
Using 'basicrapid.test' for test case definition
Using 'machine.map' for machine mapping
Runtime: 10
Measurements will be pushed to http://192.168.0.235:9091
Latency percentile measured at 99%
Connected to PROX on 10.244.1.205
Connected to PROX on 10.244.2.104
warmuptest
flowsizetest
+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------+
| UDP,    64 bytes, different number of flows by randomizing SRC & DST UDP port.

       |
+--------+-----------------+-------------+-------------+-------------+----------
----+----------+----------+----------+----------+----------+----------+-----
--+-----+
| Flows  | Speed requested  | Gen by core | Sent by NIC | Fwrd by SUT | Rec. by core
    | Avg. Lat.|99 Pcentil| Max. Lat.|    Sent    |  Received |    Lost    | Total Lost|L.Rat
io|Time|
+--------+-----------------+-------------+-------------+-------------+----------
--+-----+
|     64 | 90.0% 13.393 Mpps| 13.328 Mpps | 13.328 Mpps | 13.328 Mpps | 9.0 Gb/s | 13.328 M
pps |    12 us |    23 us |    82 us | 176446355 | 176446355 |         0 |       NA  |  0.0
0 | 11 |
+--------+-----------------+-------------+-------------+-------------+----------
----+----------+----------+----------+----------+----------+----------+-----
--+-----+
Waiting for child process 'PROX Testing on TestM1' to complete ...
Child process 'PROX Testing on TestM1' completed successfully
Child process 'PROX Testing on TestM2' completed successfully
```

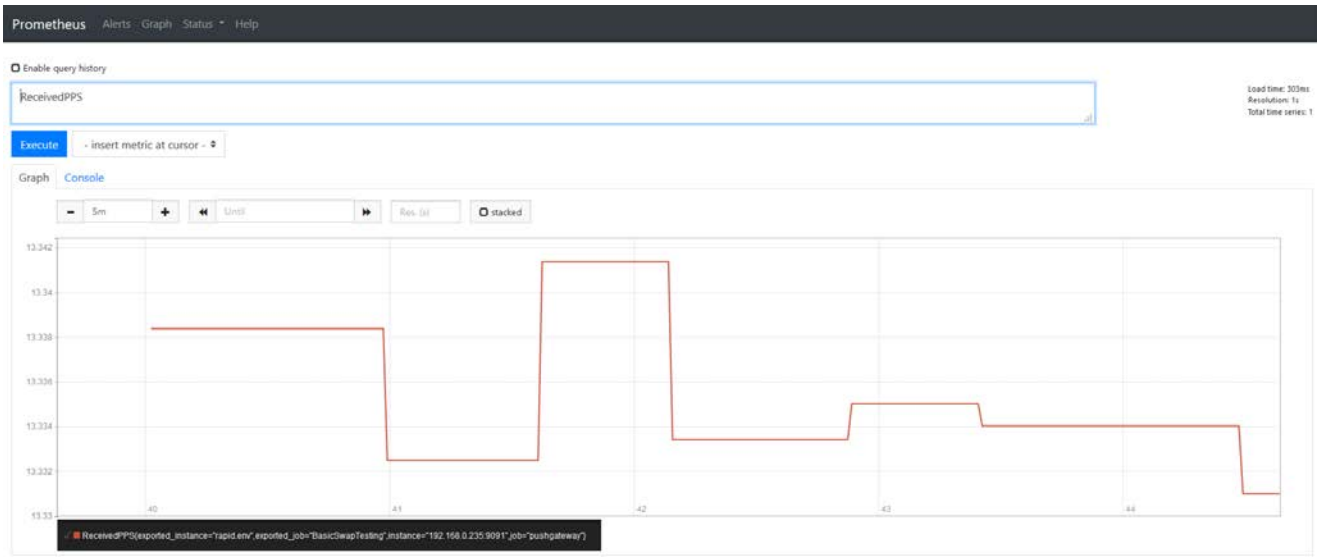
On Pushgateway (`http://$PUSHGWIP:9091`) you can see new metrics under:



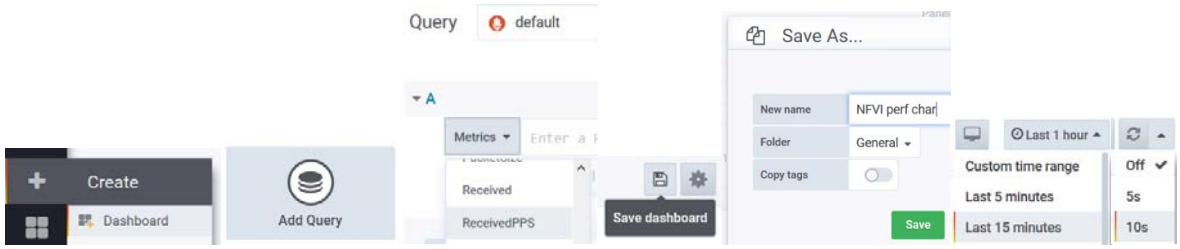
This can be visualized with Prometheus (`http://$PUSHGWIP:9090/graph`) as shown below for the metric ReceivedPPS:



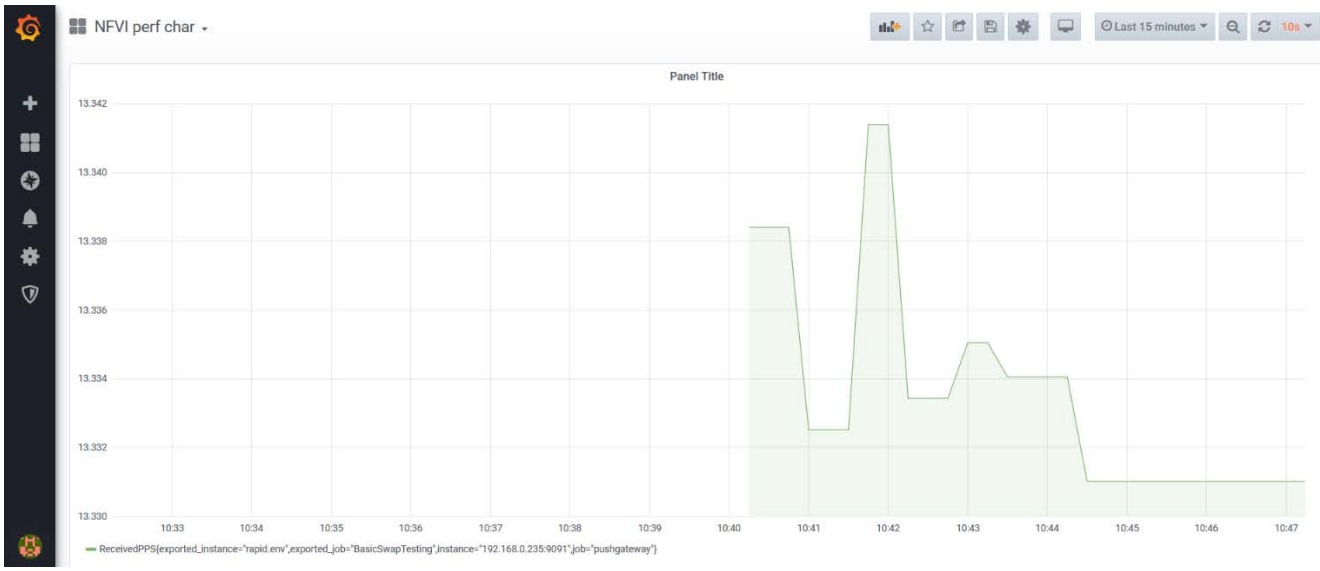
After several test runs, click on the Graph tab to see output similar to the following:



For an improved visualization, use Grafana (`http://$PUSHGWIP:3000`) after you create a new dashboard. The example below uses the Query from Prometheus using the ReceivedPPS metric, saved as "NFVI perf char", with the required refresh rate and time window:



The resulting output is similar to the following:



You can change the initial PROX speed in the `*.test` files as needed, where 100 units means 100% of 10Gbps. This is especially important if you don't have networking optimized for dataplane traffic.

After your tests are complete, you can stop running PROX pods with the command:

```
for p in ` kubectl get pod | awk ' $0~"pod-rapid" { print $1 } ' `; do kubectl delete pod $p; done
```

To further modify test cases and write new ones, refer to the online documentation and code listed in Table 2.

6 Summary

This document described the methodology and open source tools used to characterize the performance of a containerized NFVI system during the development stage. The benefits of this approach include:

- Measuring from the tenant's CNF point of view, not from a hardware generator outside the containerized environment.
- Using the same image and same engine used for various types of tests: as generator, swap/reflector or for more advanced testing (VNF/CNF Resilience Testing with PROX configured as impair gateway dropping and/or delaying packets).
- Packaging the same engine in various ways: as container bare metal, containers in VM, or a typical guest VM.
- Moving the test environment from Pets to Cattle, that is, from customized individual components to interchangeable ones.

0420/DN/PTI/PDF 621829-001US