QTIP Documentation

QTIP Team

May 09, 2019

Contents

QTIP is the project for **Platform Performance Benchmarking** in OPNFV. It aims to provide user a simple indicator for performance, simple but supported by comprehensive testing data and transparent calculation formula.

CHAPTER 1

QTIP Release Notes

1.1 Fraser

This document provides the release notes of QTIP for OPNFV Fraser release

1.1.1 Version history

Date	Ver.	Author	Comment
2018-04-25	Fraser 1.0	Zhihui Wu	

1.1.2 Summary

QTIP Fraser release supports the compute QPI(QTIP Performance Index) for **VNF**. In order to simplify the implementation, a Ubuntu 16.04 virtual machine is regarded as a simple VNF. The end users can try to run QTIP with a **real** VNF.

1.1.3 Release Data

Project	QTIP
Repo/commit-ID	qtip/opnfv-6.0.0
Release designation	stable version
Release date	2018-04-18
Purpose of the delivery	release with OPNFV cycle

Version change

Python packaging tool

Pipenv is the officially recommended Python packaging tool from Python.org.

Pipenv uses the `Pipfile` and `Pipfile.lock` instead of `requirements.txt` to manage the dependency packages.

Reason for version

Features additions

- Support the compute QPI for **VNF**

Deliverables

Software

- QTIP Docker image (tag: opnfv-6.0.0)

Documentation

- Installation & Configuration
- User Guide
- Developer Guide

1.1.4 Known Limitations, Issues and Workarounds

Limitations

N/A

Known issues

N/A

1.2 Euphrates

This document provides the release notes of QTIP for OPNFV Euphrates release

1.2.1 Version history

Date	Ver.	Author	Comment
2017-10-20	Euphrates 1.0	Yujun Zhang	

1.2.2 Summary

QTIP Euphrates release continues working on **QPI**, a.k.a. QTIP Performance Index, which is calculated from metrics collected in performance tests.

Besides compute performance benchmark, QTIP has integrated OPNFV storperf for storage performance benchmarking.

A PoC of web portal is implemented as the starting point of Benchmarking as a Service.

1.2.3 Release Data

Project	QTIP
Repo/commit-ID	qtip/euphrates.1.0
Release designation	stable version
Release date	2017-10-20
Purpose of the delivery	release with OPNFV cycle

Version change

Module version changes

The following Python packages are used in this release:

```
humanfriendly==4.4.1
connexion==1.1.11
Jinja2==2.9.6
Django==1.11.5
asq==1.2.1
six==1.11.0
ansible==2.4.0.0
requests==2.18.4
prettytable==0.7.2
numpy==1.13.1
click==6.7
pbr==3.1.1
PyYAML==3.12
```

It is considered as a baseline for future releases.

Reason for version

Features additions

- Storage QPI (QTIP Performance Index) specification and benchmarking project

Framework evolution

Ansible is used as the backbone of QTIP framework. Not only the main testing procedure is built as Ansible roles, but also the inventory discovery is implemented as Ansible module, the calculation and collection actions are Ansible plugins. Even the testing project itself is generated using jinja2 template rendering driven by Ansible.

Deliverables

Software

- QTIP Docker image (tag: euphrates.1.0)

Documentation

- Installation & Configuration
- User Guide
- Developer Guide

1.2.4 Known Limitations, Issues and Workarounds

Limitations

- Supporting on legacy OPNFV fuel installer is no longer maintained.

Known issues

1.2.5 Test Result

QTIP has undergone QA test runs with the following results:

TEST-SUITES	Results:
qtip-verify-euphrates	53/53 passed, 86% lines coverage
qtip-compute-apex-euphrates	passed
qtip-storage-apex-euphrates	passed

1.3 Danube

This document provides the release notes for Danube of QTIP.

1.3.1 Version history

Date	Ver.	Author	Comment
2017-03-30	Danube 1.0	Yujun Zhang	
2017-05-04	Danube 2.0	Yujun Zhang	
2017-07-14	Danube 3.0	Yujun Zhang	

1.3.2 Important notes

QTIP is totally reworked in Danube release. The legacy benchmarks released in Brahmaputra (compute, network and storage) are deprecated.

1.3.3 Summary

QTIP Danube release introduces **QPI**, a.k.a. QTIP Performance Index, which is calculated from metrics collected in performance tests.

A PoC of compute performance benchmark plan is provided as a sample use case.

Available benchmark plans can be listed, shown and executed from command line or over API.

1.3.4 Release Data

Project	QTIP
Repo/commit-ID	qtip/danube.3.0
Release designation	Tag update only
Release date	2017-07-14
Purpose of the delivery	OPNFV quality assurance

Version change

New in Danube 3.0

- No change in QTIP itself
- Validated on OPNFV Danube latest release

New in Danube 2.0

- Bug fix in regex of ssl

Module version changes

The following Python packages are used in this release:

```
ansible==2.1.2.0
click==6.7
connexion==1.1.5
Jinja2==2.9.5
numpy==1.12.1
paramiko==2.1.2
pbr==2.0.0
prettytable==0.7.2
six==1.10.0
PyYAML==3.12
```

It is considered as a baseline for future releases.

Reason for version

Features additions

- Compute QPI (QTIP Performance Index) specification and benchmarking plan
- Command line interface
- API server

Framework evolution

The following components are implemented and integrated

- Native runner
- File loader
- Ansible driver
- Logfile collector
- Grep parser
- Console reporter

See JIRA for full change log

Deliverables

Software

- QTIP Docker image (tag: danube.3.0)
- QTIP Docker image (tag: danube.2.0)

- QTIP Docker image (tag: danube.1.0)

Documentation

- Installation & Configuration
- User Guide
- Developer Guide

1.3.5 Known Limitations, Issues and Workarounds

Limitations

- The compute benchmark plan is hard coded in native runner
- Baseline for Compute QPI is not created yet, therefore scores are not available

Known issues

- QTIP-230 - logger warns about socket /dev/log when running in container

1.3.6 Test Result

QTIP has undergone QA test runs with the following results:

TEST-SUITES	Results:
qtip-verify-danube	94/94 passed
qtip-os-nosdn-kvm-ha-zte-pod3-daily-danube	passed
qtip-os-nosdn-nofeature-ha-zte-pod3-daily-danube	passed
qtip-os-odl_l2-nofeature-ha-zte-pod1-daily-danube	passed

1.4 Brahmaputra

NOTE: The release note for OPNFV Brahmaputra is missing. This is a copy of the README.

1.4.1 QTIP Benchmark Suite

QTIP is a benchmarking suite intended to benchmark the following components of the OPNFV Platform:

1. Computing components
2. Networking components
3. Storage components

The efforts in QTIP are mostly focused on identifying

1. Benchmarks to run
2. Test cases in which these benchmarks to run

3. Automation of suite to run benchmarks within different test cases
4. Collection of test results

QTIP Framework can now be called: (qtip.py).

The Framework can run 5 computing benchmarks:

1. Dhrystone
2. Whetstone
3. RamBandwidth
4. SSL
5. nDPI

These benchmarks can be run in 2 test cases:

1. VM vs Baremetal
2. Baremetal vs Baremetal

Instructions to run the script:

1. Download and source the OpenStack *adminrc* file for the deployment on which you want to create the VM for benchmarking
2. run *python qtip.py -s {SUITE} -b {BENCHMARK}*
3. run *python qtip.py -h* for more help
4. list of benchmarks can be found in the *qtip/test_cases* directory
5. SUITE refers to compute, network or storage

Requirements:

1. Ansible 1.9.2
2. Python 2.7
3. PyYAML

Configuring Test Cases:

Test cases can be found within the *test_cases* directory. For each Test case, a Config.yaml file contains the details for the machines upon which the benchmarks would run. Edit the IP and the Password fields within the files for the machines on which the benchmark is to run. A robust framework that would allow to include more tests would be included within the future.

Jump Host requirements:

The following packages should be installed on the server from which you intend to run QTIP.

1: Heat Client 2: Glance Client 3: Nova Client 4: Neutron Client 5: wget 6: PyYaml

Networking

1: The Host Machines/compute nodes to be benchmarked should have public/access network 2: The Host Machines/compute nodes should allow Password Login

QTIP support for Foreman

{TBA}

CHAPTER 2

QTIP Installation Guide

2.1 Configuration

QTIP currently supports by using a Docker image. Detailed steps about setting up QTIP can be found below.

To use QTIP you should have access to an OpenStack environment, with at least Nova, Neutron, Glance, Keystone and Heat installed. Add a brief introduction to configure OPNFV with this specific installer

2.1.1 Installing QTIP using Docker

QTIP docker image

QTIP has a Docker images on the docker hub. Pulling opnfv/qtip docker image from docker hub:

```
docker pull opnfv/qtip:stable
```

Verify that `opnfv/qtip` has been downloaded. It should be listed as an image by running the following command.

```
docker images
```

Run and enter the docker instance

1. If you want to run benchmarks:

```
envs="INSTALLER_TYPE={INSTALLER_TYPE} -e INSTALLER_IP={INSTALLER_IP} -e NODE_NAME=
↪{NODE_NAME}"
docker run -p [HOST_IP:]<HOST_PORT>:5000 --name qtip -id -e $envs opnfv/qtip
docker start qtip
docker exec -i -t qtip /bin/bash
```

INSTALLER_TYPE should be one of OPNFV installer, e.g. apex, compass, daisy, fuel and joid. Currenty, QTIP only supports installer fuel.

INSTALLER_IP is the ip address of the installer that can be accessed by QTIP.

NODE_NAME is the name of opnfv pod, e.g. zte-pod1.

2. If you do not want to run any benchmarks:

```
docker run --name qtip -id opnfv/qtip
docker exec -i -t qtip /bin/bash
```

Now you are in the container and QTIP can be found in the /repos/qtip and can be navigated to using the following command.

```
cd repos/qtip
```

2.1.2 Install from source code

You may try out the latest version of QTIP by installing from source code. It is recommended to run it under Python virtualenv so it won't screw system libraries.

Run the following commands:

```
git clone https://git.opnfv.org/qtip && cd qtip
virtualenv .venv && source .venv/bin/activate
pip install -e .
```

Use the following command to exit virtualenv:

```
deactivate
```

Re-enter the virtualenv with:

```
cd <qtip-directory>
source .venv/bin/activate
```

2.1.3 Environment configuration

Hardware configuration

QTIP does not have specific hardware requirements, and it can runs over any OPNFV installer.

Jumphost configuration

Installer Docker on Jumphost, which is used for running QTIP image.

You can refer to these links:

Ubuntu: https://docs.docker.com/engine/installation/linux/ubuntu/

Centos: https://docs.docker.com/engine/installation/linux/centos/

Platform components configuration

Describe the configuration of each component in the installer.

CHAPTER 3

QTIP User Guide

3.1 Overview

QTIP is the project for **Platform Performance Benchmarking** in OPNFV. It aims to provide user a simple indicator for performance, simple but supported by comprehensive testing data and transparent calculation formula.

QTIP introduces a concept called **QPI**, a.k.a. QTIP Performance Index, which aims to be a **TRUE** indicator of performance. **TRUE** reflects the core value of QPI in four aspects

- *Transparent*: being an open source project, user can inspect all details behind QPI, e.g. formulas, metrics, raw data
- *Reliable*: the integrity of QPI will be guaranteed by traceability in each step back to raw test result
- *Understandable*: QPI is broke down into section scores, and workload scores in report to help user to understand
- *Extensible*: users may create their own QPI by composing the existed metrics in QTIP or extend new metrics

3.1.1 Benchmarks

The builtin benchmarks of QTIP are located in `<package_root>/benchmarks` folder

- *QPI*: specifications about how an QPI is calculated and sources of metrics
- *metric*: performance metrics referred in QPI, currently it is categorized by performance testing tools
- *plan*: executable benchmarking plan which collects metrics and calculate QPI

3.2 Getting started with QTIP

3.2.1 Installation

Refer to installation and configuration guide for details

3.2.2 Create

Create a new project to hold the necessary configurations and test results

```
qtip create <project_name>
```

The user would be prompted for OPNFV installer, its hostname etc

```
**Pod Name [unknown]: zte-pod1**
User's choice to name OPNFV Pod

**OPNFV Installer [manual]: fuel**
QTIP currently supports fuel and apex only

**Installer Hostname [dummy-host]: master**
The hostname for the fuel or apex installer node. The same hostname can be added to␣
↪**~/.ssh/config** file of current user,
if there are problems resolving the hostname via interactive input.

**OPNFV Scenario [unknown]: os-nosdn-nofeature-ha**
Depends on the OPNFV scenario deployed
```

3.2.3 Setup

With the project is created, user should now proceed on to setting up testing environment. In this step, ssh connection to hosts in SUT will be configured automatically:

```
cd <project_name>
$ qtip setup
```

3.2.4 Run

QTIP uses `ssh-agent` for authentication of ssh connection to hosts in SUT. It must be started correctly before running the tests:

```
eval $(ssh-agent)
```

Then run test with `qtip run`

3.2.5 Teardown

Clean up the temporary folder on target hosts.

Note: The installed packages for testing won't be uninstalled.

3.2.6 One more thing

You may use `-v` for verbose output (`-vvv` for more, `-vvvv` to enable connection debugging)

3.3 CLI User Manual

QTIP consists of a number of benchmarking tools or metrics, grouped under QPI's. QPI's map to the different components of a NFVi ecosystem, such as compute, network and storage. Depending on the type of application, a user may group them under plans.

3.3.1 Bash Command Completion

To enable command completion, an environment variable needs to be enabled. Add the following line to the **.bashrc** file

```
eval "$(_QTIP_COMPLETE=source qtip)"
```

3.3.2 Getting help

QTIP CLI provides interface to all of the above the components. A help page provides a list of all the commands along with a short description.

```
qtip --help
```

3.3.3 Usage

QTIP is currently supports two different QPI's, compute and storage. To list all the supported QPI

```
qtip qpi list
```

The details of any QPI can be viewed as follows

qtip qpi show <qpi_name>

In order to benchmark either one of them, their respective templates need to be generated

```
qtip create --project-template [compute|storage] <workspace_name>
```

By default, the compute template will be generated. An interactive prompt would gather all parameters specific to OpenStack installation.

Once the template generation is complete, configuration for OpenStack needs to be generated.

```
cd <workspace_name>
qtip setup
```

This step generates the inventory, populating it with target nodes.

QTIP can now be run

```
qtip run
```

This would start the complete testing suite, which is either compute or storage. Each suite normally takes about half an hour to complete.

Benchmarking report is made for each and every individual section in a QPI, on a particular target node. It consists of the actual test values on that node along with scores calculated by comparison against a baseline.

```
qtip report show [-n|--node] <node> <section_name>
```

3.3.4 Debugging options

QTIP uses Ansible as the runner. One can use all of Ansible's CLI option with QTIP. In order to enable verbose mode

```
qtip setup -v
```

One may also be able to achieve the different levels of verbosity

```
qtip run [-v|-vv|-vvv]
```

3.4 API User Manual

QTIP consists of a number of benchmarking tools or metrics, grouped under QPI's. QPI's map to the different components of an NFVI ecosystem, such as compute, network and storage. Depending on the type of application, a user may group them under plans.

QTIP API provides a RESTful interface to all of the above components. User can retrieve list of plans, QPIs and metrics and their individual information.

3.4.1 Running

After installing QTIP. API server can be run using command `qtip-api` on the local machine.

All the resources and their corresponding operation details can be seen at `/v1.0/ui`.

The whole API specification in json format can be seen at `/v1.0/swagger.json`.

The data models are given below:

- Plan
- Metric
- QPI

Plan:

```
{
  "name": <plan name>,
  "description": <plan profile>,
  "info": <{plan info}>,
  "config": <{plan configuration}>,
  "QPIs": <[list of qpis]>,
},
```

Metric:

```
{
  "name": <metric name>,
  "description": <metric description>,
  "links": <[links with metric information]>,
  "workloads": <[cpu workloads(single_cpu, multi_cpu]>,
},
```

QPI:

```
{
  "name": <qpi name>,
  "description": <qpi description>,
  "formula": <formula>,
  "sections": <[list of sections with different metrics and formulaes]>,
}
```

The API can be described as follows

Plans:

Method	Path	Description
GET	/v1.0/plans	Get the list of of all plans
GET	/v1.0/plans/{name}	Get details of the specified plan

Metrics:

Method	Path	Description
GET	/v1.0/metrics	Get the list of all metrics
GET	/v1.0/metrics/{name}	Get details of specified metric

QPIs:

Method	Path	Description
GET	/v1.0/qpis	Get the list of all QPIs
GET	/v1.0/qpis/{name}	Get details of specified QPI

Note: *running API with connexion cli does not require base path (/v1.0/) in url*

3.5 Compute Performance Benchmarking

The compute QPI aims to benchmark the compute components of an OPNFV platform. Such components include, the CPU performance, the memory performance.

The compute QPI consists of both synthetic and application specific benchmarks to test compute components.

All the compute benchmarks could be run in the scenario: On Baremetal Machines provisioned by an OPNFV installer (Host machines) On Virtual machines provisioned by OpenStack deployed by an OPNFV installer

Note: The Compute benchmank constains relatively old benchmarks such as dhrystone and whetstone. The suite would be updated for better benchmarks such as Linbench for the OPNFV future release.

3.5.1 Getting started

Notice: All descriptions are based on QTIP container.

Inventory File

QTIP uses Ansible to trigger benchmark test. Ansible uses an inventory file to determine what hosts to work against. QTIP can automatically generate a inventory file via OPNFV installer. Users also can write their own inventory

information into `/home/opnfv/qtip/hosts`. This file is just a text file containing a list of host IP addresses. For example:

```
[hosts]
10.20.0.11
10.20.0.12
```

QTIP key Pair

QTIP use a SSH key pair to connect to remote hosts. When users execute compute QPI, QTIP will generate a key pair named *QtipKey* under `/home/opnfv/qtip/` and pass public key to remote hosts.

If environment variable *CI_DEBUG* is set to *true*, users should delete it by manual. If *CI_DEBUG* is not set or set to *false*, QTIP will delete the key from remote hosts before the execution ends. Please make sure the key deleted from remote hosts or it can introduce a security flaw.

Execution

There are two ways to execute compute QPI:

- Script

  You can run compute QPI with docker exec:

  ```
  # run with baremetal machines provisioned by an OPNFV installer
  docker exec <qtip container> bash -x /home/opnfv/repos/qtip/qtip/scripts/
  ↪quickstart.sh -q compute

  # run with virtual machines provisioned by OpenStack
  docker exec <qtip container> bash -x /home/opnfv/repos/qtip/qtip/scripts/
  ↪quickstart.sh -q compute -u vnf
  ```

- Commands

  In a QTIP container, you can run compute QPI by using QTIP CLI. You can get more details from *userguide/cli.rst*.

Test result

QTIP generates results in the `/home/opnfv/<project_name>/results/` directory are listed down under the timestamp name.

Metrics

The benchmarks include:

Dhrystone 2.1

Dhrystone is a synthetic benchmark for measuring CPU performance. It uses integer calculations to evaluate CPU capabilities. Both Single CPU performance is measured along multi-cpu performance.

Dhrystone, however, is a dated benchmark and has some short comings. Written in C, it is a small program that doesn't test the CPU memory subsystem. Additionally, dhrystone results could be modified by optimizing the compiler and insome cases hardware configuration.

References: http://www.eembc.org/techlit/datasheets/dhrystone_wp.pdf

Whetstone

Whetstone is a synthetic benchmark to measure CPU floating point operation performance. Both Single CPU performance is measured along multi-cpu performance.

Like Dhrystone, Whetstone is a dated benchmark and has short comings.

References:

http://www.netlib.org/benchmark/whetstone.c

OpenSSL Speed

OpenSSL Speed can be used to benchmark compute performance of a machine. In QTIP, two OpenSSL Speed benchmarks are incorporated:

1. RSA signatunes/sec signed by a machine
2. AES 128-bit encryption throughput for a machine for cipher block sizes

References:

https://www.openssl.org/docs/manmaster/apps/speed.html

RAMSpeed

RAMSpeed is used to measure a machine's memory perfomace. The problem(array)size is large enough to ensure Cache Misses so that the main machine memory is used.

INTmem and FLOATmem benchmarks are executed in 4 different scenarios:

a. Copy: a(i)=b(i)

b. Add: a(i)=b(i)+c(i)

c. Scale: a(i)=b(i)*d

d. Tniad: a(i)=b(i)+c(i)*d

INTmem uses integers in these four benchmarks whereas FLOATmem uses floating points for these benchmarks.

References:

http://alasir.com/software/ramspeed/

https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W51a7ffcf4dfd_4b40_9d82_446ebc23c550/page/Untangling+memory+access+measurements

DPI

nDPI is a modified variant of OpenDPI, Open source Deep packet Inspection, that is maintained by ntop. An example application called *pcapreader* has been developed and is available for use along nDPI.

A sample .pcap file is passed to the *pcapreader* application. nDPI classifies traffic in the pcap file into different categories based on string matching. The *pcapreader* application provides a throughput number for the rate at which

traffic was classified, indicating a machine's computational performance. The results are run 10 times and an average is taken for the obtained number.

nDPI may provide non consistent results and was added to Brahmaputra for experimental purposes

References:

http://www.ntop.org/products/deep-packet-inspection/ndpi/

http://www.ntop.org/wp-content/uploads/2013/12/nDPI_QuickStartGuide.pdf

3.6 Storage Performance Benchmarking

Like compute QPI, storage QPI gives users an overall score for system storage performance. The project StorPerf in OPNFV provides a tool to measure ephemeral and block storage performance of OpenStack. Naturally, QTIP integrates StorPerf to generate the storage performance data.

For now, storage QPI runs against on the baremetal/virtual scenario deployed by the OPNFV installer APEX.

3.6.1 Getting started

Notice: All descriptions are based on containers.

Requirements

- Git must be installed.
- Docker and docker-compose must be installed.

Git Clone QTIP Repo

```
git clone https://git.opnfv.org/qtip
```

Running QTIP container and Storperf Containers

With Docker Compose, we can use a YAML file to configure application's services and use a single command to create and start all the services.

There is a YAML file `./qtip/tests/ci/storage/docker-compose.yaml` from QTIP repos. It can help you to create and start the storage QPI service.

Before running docker-compose, you must specify these three variables:

- DOCKER_TAG, which specified the Docker tag(ie: latest)
- SSH_CREDENTIALS, a directory which includes an SSH key pair will be mounted into QTIP container. QTIP use this SSH key pair to connect to remote hosts.
- ENV_FILE, which includes the environment variables required by QTIP and Storperf containers

  A example of ENV_FILE:

```
INSTALLER_TYPE=apex
INSTALLER_IP=192.168.122.247
TEST_SUITE=storage
NODE_NAME=zte-virtual5
SCENARIO=generic
TESTAPI_URL=
OPNFV_RELEASE=euphrates
# The below environment variables are Openstack Credentials.
OS_USERNAME=admin
OS_USER_DOMAIN_NAME=Default
OS_PROJECT_DOMAIN_NAME=Default
OS_BAREMETAL_API_VERSION=1.29
NOVA_VERSION=1.1
OS_PROJECT_NAME=admin
OS_PASSWORD=ZjmZJmkCvVXf9ry9daxgwmz3s
OS_NO_CACHE=True
COMPUTE_API_VERSION=1.1
no_proxy=,192.168.37.10,192.0.2.5
OS_CLOUDNAME=overcloud
OS_AUTH_URL=http://192.168.37.10:5000/v3
IRONIC_API_VERSION=1.29
OS_IDENTITY_API_VERSION=3
OS_AUTH_TYPE=password
```

Then, you use the following commands to start storage QPI service.

```
docker-compose -f docker-compose.yaml pull
docker-compose -f docker-compose.yaml up -d
```

Execution

- Script

  You can run storage QPI with docker exec:

  ```
  docker exec <qtip container> bash -x /home/opnfv/repos/qtip/qtip/scripts/
  ↪quickstart.sh
  ```

- Commands

  In a QTIP container, you can run storage QPI by using QTIP CLI. You can get more details from *userguide/cli.rst*.

Test result

QTIP generates results in the `/home/opnfv/<project_name>/results/` directory are listed down under the timestamp name.

Metrics

Storperf provides the following metrics:

- IOPS
- Bandwidth (number of kilobytes read or written per second)

- Latency

3.7 Network Performance Benchmarking

Like compute or storage QPI, network QPI gives users an overall score for system network performance. For now it focuses on L2 virtual switch performance on NFVI. Current testcase are from RFC2544 standart and implemntation is based on Spirent Testcenter Virtual.

For now, network QPI runs against on the baremetal/virtual scenario deployed by the OPNFV installer APEX.

3.7.1 Getting started

Notice: All descriptions are based on containers.

Requirements

- Git must be installed.
- Docker and docker-compose must be installed.
- Spirent Testcenter Virtual image must be uploaded to the target cloud and the associated flavor must be created before test.
- Spirent License Server and Spirent LabServer must be set up and keep them ip reachable from target cloud external network before test.

Git Clone QTIP Repo

```
git clone https://git.opnfv.org/qtip
```

Running QTIP container and Nettest Containers

With Docker Compose, we can use a YAML file to configure application's services and use a single command to create and start all the services.

There is a YAML file `./qtip/tests/ci/network/docker-compose.yaml` from QTIP repos. It can help you to create and start the network QPI service.

Before running docker-compose, you must specify these three variables:

- DOCKER_TAG, which specified the Docker tag(ie: latest)
- SSH_CREDENTIALS, a directory which includes an SSH key pair will be mounted into QTIP container. QTIP use this SSH key pair to connect to remote hosts.
- ENV_FILE, which includes the environment variables required by QTIP and Storperf containers

  A example of ENV_FILE:

```
INSTALLER_TYPE=apex
INSTALLER_IP=192.168.122.247
TEST_SUITE=network
NODE_NAME=zte-virtual5
SCENARIO=generic
TESTAPI_URL=
OPNFV_RELEASE=euphrates
# The below environment variables are Openstack Credentials.
OS_USERNAME=admin
OS_USER_DOMAIN_NAME=Default
OS_PROJECT_DOMAIN_NAME=Default
OS_BAREMETAL_API_VERSION=1.29
NOVA_VERSION=1.1
OS_PROJECT_NAME=admin
OS_PASSWORD=ZjmZJmkCvVXf9ry9daxgwmz3s
OS_NO_CACHE=True
COMPUTE_API_VERSION=1.1
no_proxy=,192.168.37.10,192.0.2.5
OS_CLOUDNAME=overcloud
OS_AUTH_URL=http://192.168.37.10:5000/v3
IRONIC_API_VERSION=1.29
OS_IDENTITY_API_VERSION=3
OS_AUTH_TYPE=password
# The below environment variables are extra info with Spirent.
SPT_LICENSE_SERVER_IP=192.168.37.251
SPT_LAB_SERVER_IP=192.168.37.122
SPT_STCV_IMAGE_NAME=stcv-4.79
SPT_STCV_FLAVOR_NAME=m1.tiny
```

Then, you use the following commands to start network QPI service.

```
docker-compose -f docker-compose.yaml pull
docker-compose -f docker-compose.yaml up -d
```

Execution

You can run network QPI with docker exec:

```
docker exec <qtip container> bash -x /home/opnfv/repos/qtip/qtip/scripts/quickstart.sh
```

QTIP generates results in the `$PWD/results/` directory are listed down under the timestamp name.

Metrics

Nettest provides the following metrics:

- RFC2544 througput
- RFC2544 latency

3.8 Test Case Description

Network throughput			
test case id	qtip_throughput		
metric	rfc2544 throughput		
test purpose	get the max throughput of the pathway on same host or accross hosts		
configuration	None		
test tool	Spirent Test Center Virtual		
references	RFC2544		
applicability	1. test the switch throughput on same host or accross hosts 2. test the switch throughput for different packet sizes		
pre-test conditions	1. deploy STC license server and LabServer on public network and verify it can operate correctlly 2. upload STC virtual image and create STCv flavor on the deployed cloud environment		
test sequence	step	description	result
	1	deploy STCv stack on the target cloud with affinity attribute according to requirements.	2 STCv VM will be established on the cloud
	2	run rfc2544 throughput test with different packet size	test result report will be produced in QTIP container
	3	destory STCv stack different packet size	STCv stack destoried
test verdict	find the test result report in QTIP container running directory		

Network throughput			
test case id	qtip_latency		
metric	rfc2544 lantency		
test purpose	get the latency value of the pathway on same host or accross hosts		
configuration	None		
test tool	Spirent Test Center Virtual		
references	RFC2544		
applicability	1. test the switch latency on same host or accross hosts 2. test the switch latency for different packet sizes		
pre-test conditions	1. deploy STC license server and LabServer on public network and verify it can operate correctlly 2. upload STC virtual image and create STCv flavor on the deployed cloud environment		
test sequence	step	description	result
	1	deploy STCv stack on the target cloud with affinity attribute according to requirements.	2 STCv VM will be established on the cloud
	2	run rfc2544 latency test with different packet size	test result report will be produced in QTIP container
	3	destroy STCv stack	STCv stack destried
test verdict	find the test result report in QTIP container running directory		

CHAPTER 4

QTIP Developer Guide

4.1 Overview

QTIP uses Python as primary programming language and build the framework from the following packages

Module	Package
api	Connexion - API first applications with OpenAPI/Swagger and Flask
cli	Click - the "Command Line Interface Creation Kit"
template	Jinja2 - a full featured template engine for Python
docs	sphinx - a tool that makes it easy to create intelligent and beautiful documentation
testing	pytest - a mature full-featured Python testing tool that helps you write better programs

4.1.1 Source Code

The structure of repository is based on the recommended sample in The Hitchhiker's Guide to Python

Path	Content
`./benchmarks/`	builtin benchmark assets including plan, QPI and metrics
`./contrib/`	independent project/plugin/code contributed to QTIP
`./docker/`	configuration for building Docker image for QTIP deployment
`./docs/`	release notes, user and developer documentation, design proposals
`./legacy/`	legacy obsoleted code that is unmaintained but kept for reference
`./opt/`	optional component, e.g. scripts to setup infrastructure services for QTIP
`./qtip/`	the actual package
`./tests/`	package functional and unit tests
`./third-party/`	third part included in QTIP project

4.1.2 Coding Style

QTIP follows OpenStack Style Guidelines for source code and commit message.

Specially, it is recommended to link each patch set with a JIRA issue. Put:

```
JIRA: QTIP-n
```

in commit message to create an automatic link.

4.1.3 Testing

All testing related code are stored in `./tests/`

Path	Content
`./tests/data/`	data fixtures for testing
`./tests/unit/`	unit test for each module, follow the same layout as ./qtip/
`./conftest.py`	pytest configuration in project scope

tox is used to automate the testing tasks

```
cd <project_root>
pip install tox
tox
```

The test cases are written in pytest. You may run it selectively with

```
pytest tests/unit/reporter
```

4.1.4 Branching

Stable branches are created when features are frozen for next release. According to OPNFV release milestone description, stable branch window is open on MS6 and closed on MS7.

1. Contact gerrit admin <opnfv-helpdesk@rt.linuxfoundation.org> to create branch for project.
2. Setup qtip jobs and docker jobs for stable branch in releng
3. Follow instructions for stable branch.

NOTE: we do **NOT** create branches for feature development as in the popular GitHub Flow

4.1.5 Releasing

Tag Deliverable and write release note

Git repository

Follow the example in Git Tagging Instructions for Danube to tag the source code:

```
git fetch gerrit
git checkout stable/<release-name>
git tag -am "<release-version>" <release-version>
git push gerrit <release-version>
```

Docker image

1. Login OPNFV Jenkins
2. Go to the **'qtip-docker-build-push-<release>'_** and click "Build With Parameters"
3. Fill in `RELEASE_VERSION` with version number not including release name, e.g. `1.0`
4. Trigger a manual build

Python Package

QTIP is also available as a Python Package. It is hosted on the Python Package Index(PyPI).

1. Install twine with `pip install twine`
2. Build the distributions `python setup.py sdist bdist_wheel`
3. Upload the distributions built with `twine upload dist/*`

NOTE: only package **maintainers** are permitted to upload the package versions.

Release note

Create release note under `qtip/docs/release/release-notes` and update `index.rst`

4.2 Run with Ansible

QTIP benchmarking tasks are built upon Ansible playbooks and roles. If you are familiar with Ansible, it is possible to run it with `ansible-playbook` command. And it is useful during development of ansible modules or testing roles.

4.2.1 Create workspace

There is a playbook in `resources/ansible_roles/qtip-workspace` used for creating a new workspace:

```
cd resources/ansible_roles/qtip-workspace
ansible-playbook create.yml
```

NOTE: if this playbook is moved to other directory, configuration in `ansible.cfg` needs to be updated accordingly. The ansible roles from QTIP, i.e. `<path_of_qtip>/resources/ansible_roles` must be added to `roles_path` in *Ansible configuration file*. For example:

```
roles_path = ~/qtip/resources/ansible_roles
```

4.2.2 Executing benchmark

Before executing the setup playbook, make sure *~/.ssh/config* has been configured properly so that you can login the **master node** "directly". Skip next section, if you can login with `ssh <master-host>` from localhost,

SSH access to master node

It is common that the master node is behind some jump host. In this case, ssh option `ProxyCommand` and `ssh-agent` shall be required.

Assume that you need to login to deploy server, then login to the master node from there. An example configuration is as following:

```
Host fuel-deploy
  HostName 172.50.0.250
  User root

Host fuel-master
  HostName 192.168.122.63
  User root
  ProxyCommand ssh -o 'ForwardAgent yes' apex-deploy 'ssh-add && nc %h %p'
```

If several jumps are required to reach the master node, we may chain the jump hosts like below:

```
Host jumphost
  HostName 10.62.105.31
  User zte
  Port 22

Host fuel-deploy
  HostName 172.50.0.250
  User root
  ProxyJump jumphost

Host fuel-master
  HostName 192.168.122.63
  User root
  ProxyCommand ssh -o 'ForwardAgent yes' apex-deploy 'ssh-add && nc %h %p'
```

NOTE: `ProxyJump` is equivalent to the long `ProxyCommand` option, but it is only available since OpenSSH 7.3

Automatic setup

1. Modify `<workspace>/group_vars/all.yml` to set installer information correctly
2. Modify `<workspace>/hosts` file to set installer master host correctly

#. Run the setup playbook to generate ansible inventory of system under test by querying the slave nodes from the installer master:

```
cd workspace
ansible-playbook setup.yml
```

It will update the `hosts` and `ssh.cfg`

Currently, QTIP supports automatic discovery from apex and fuel.

Manual setup

If your installer is not supported or you are testing hosts not managed by installer, you may add them manually in `[compute]` group in `<workspace>/hosts`:

```
[compute:vars]
ansible_ssh_common_args=-F ./ssh.cfg

[compute]
node-2
node-4
node-6
node-7
```

And `ssh.cfg` for ssh connection configuration:

```
Host node-5
  HostName 10.20.5.12
  User root
```

Run the tests

Run the benchmarks with the following command:

```
ansible-playbook run.yml
```

CAVEAT: QTIP will install required packages in system under test.

Inspect the results

The test results and calculated output are stored in `results`:

```
current/
    node-2/
        arithmetic/
            metric.json
            report
            unixbench.log
        dpi/
        ...
    node-4/
    ...
    qtip-pod-qpi.json
qtip-pod-20170425-1710/
qtip-pod-20170425-1914/
...
```

The folders are named as `<pod_name>-<start_time>/` and the results are organized by *hosts* under test. Inside each host, the test data are organized by metrics as defined in QPI specification.

For each metrics, it usually includes the following content

- log file generated by the performance testing tool
- metrics collected from the log files
- reported rendered with the metrics collected

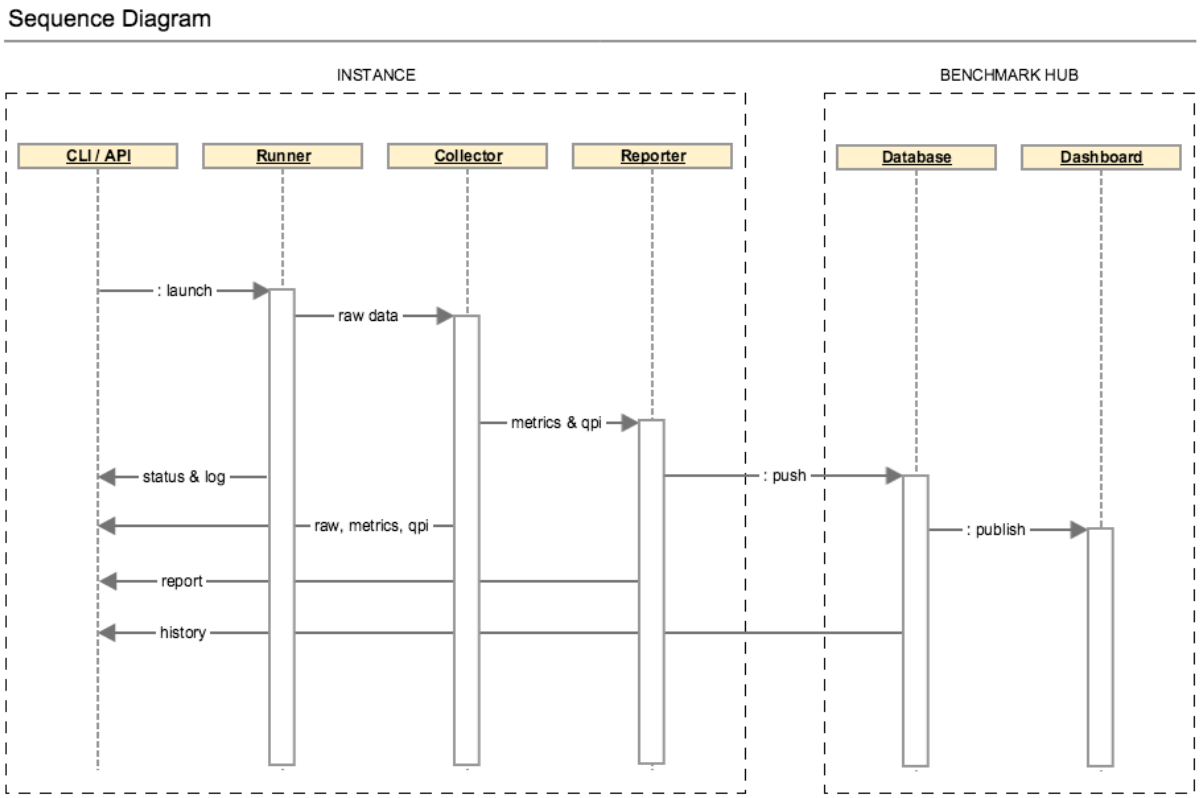
Teardown the test environment

QTIP will create temporary files for testing in system under test. Execute the teardown playbook to clean it up:

```
ansible-playbook teardown.yml
```

4.3 Architecture

In Danube, QTIP releases its standalone mode, which is also know as `solo`:

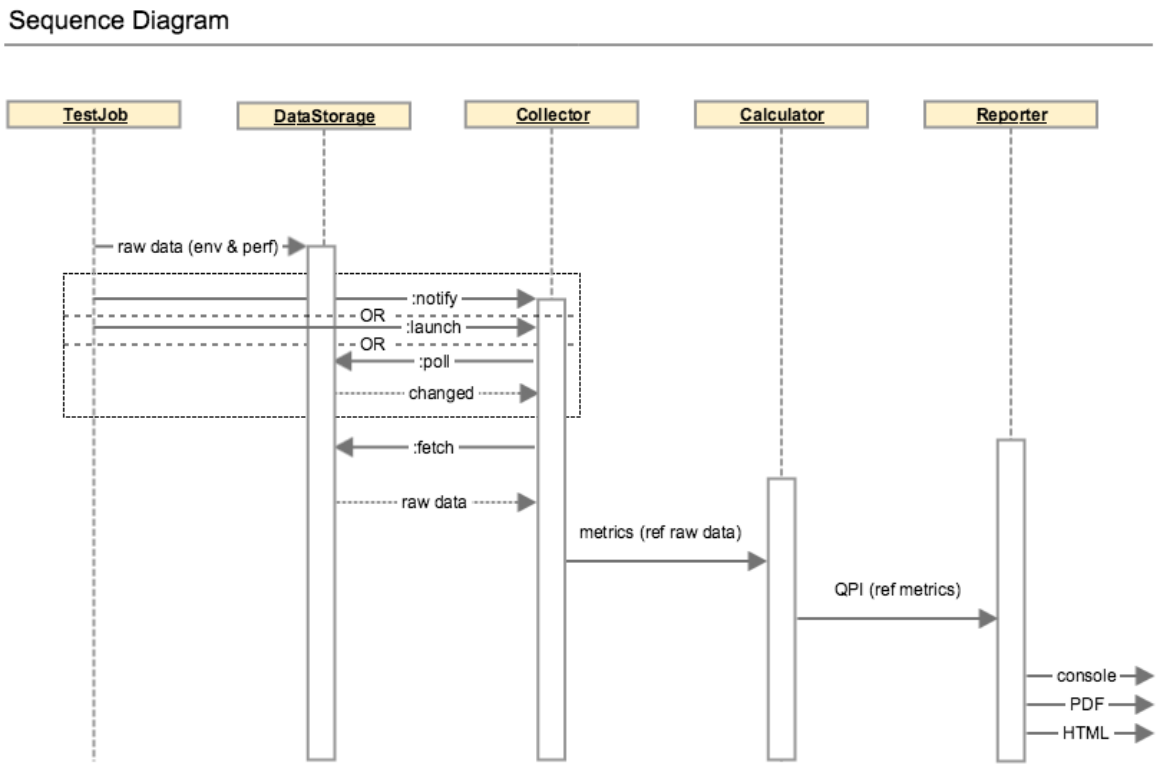


The runner could be launched from CLI (command line interpreter) or API (application programming interface) and drives the testing jobs. The generated data including raw performance data and testing environment are fed to collector. Performance metrics will be parsed from the raw data and used for QPI calculation. Then the benchmark report is rendered with the benchmarking results.

The execution can be detailed in the diagram below:

4.4 Framework

QTIP is built upon Ansible by extending modules, playbook roles and plugins.



4.4.1 Modules

QTIP creates dedicated modules to gather slave node list and information from installer master. See embedded document in `qtip/ansible_library/modules` for details

4.4.2 Plugins

Stored in `qtip/ansible_library/plugins`

Action plugins

Several action plugins have been created for test data post processing

- collect - parse and collect metrics from raw test results like log files
- calculate - calculate score according to specification
- aggregate - aggregate calculated results from all hosts under test

4.4.3 Playbook roles

QTIP roles

- qtip - main qtip tasks
- qtip-common - common tasks required in QTIP

- qtip-workspace - generate a workspace for running benchmarks

`qtip` roles should be included with a specified `action` and `output` directory, e.g.:

```
- { role: inxi, output: "{{ qtip_results }}/sysinfo", tags: [run, inxi, sysinfo] }
```

testing roles

Testing roles are organized by testing tools

- inxi - system information tool
- nDPI
- openssl
- ramspeed
- unixbench

supporting roles

- opnfv-testapi - report result to testapi

4.4.4 Tags

Tags are used to categorize the test tasks from different aspects.

- stages like `run`, `collect`, `calculate`, `aggregate`, `report`
- test tools like `inxi`, `ndpi` and etc
- information or metrics like `sysinfo`, `dpi`, `ssl`

Use

- `ansible-playbook run.yml --list-tags` to list all tags
- `ansible-playbook run.yml --list-tasks` to list all tasks

During development of post processing, you may skip `run` stage to save time, e.g. `ansible-playbook run.yml --tags collect,calculate,aggregate`

4.5 CLI - Command Line Interface

QTIP consists of different tools(metrics) to benchmark the NFVI. These metrics fall under different NFVI subsystems(QPI's) such as compute, storage and network. A plan consists of one or more QPI's, depending upon how the end user would want to measure performance. CLI is designed to help the user, execute benchmarks and view respective scores.

4.5.1 Framework

QTIP CLI has been created using the Python package Click, Command Line Interface Creation Kit. It has been chosen for number of reasons. It presents the user with a very simple yet powerful API to build complex applications. One of the most striking features is command nesting.

As explained, QTIP consists of metrics, QPI's and plans. CLI is designed to provide interface to all these components. It is responsible for execution, as well as provide listing and details of each individual element making up these components.

4.5.2 Design

CLI's entry point extends Click's built in MultiCommand class object. It provides two methods, which are overridden to provide custom configurations.

```
class QtipCli(click.MultiCommand):

    def list_commands(self, ctx):
        rv = []
        for filename in os.listdir(cmd_folder):
            if filename.endswith('.py') and \
                filename.startswith('cmd_'):
                rv.append(filename[4:-3])
        rv.sort()
        return rv

    def get_command(self, ctx, name):
        try:
            if sys.version_info[0] == 2:
                name = name.encode('ascii', 'replace')
            mod = __import__('qtip.cli.commands.cmd_' + name,
                             None, None, ['cli'])
        except ImportError:
            return
        return mod.cli
```

Commands and subcommands will then be loaded by the `get_command` method above.

4.5.3 Extending the Framework

Framework can be easily extended, as per the users requirements. One such example can be to override the builtin configurations with user defined ones. These can be written in a file, loaded via a Click Context and passed through to all the commands.

```
class Context:

    def __init__():

        self.config = ConfigParser.ConfigParser()
        self.config.read('path/to/configuration_file')

    def get_paths():

        paths = self.config.get('section', 'path')
        return paths
```

The above example loads configuration from user defined paths, which then need to be provided to the actual command definitions.

```
from qtip.cli.entry import Context

```

(continues on next page)

(continued from previous page)

```
pass_context = click.make_pass_decorator(Context, ensure=False)

@cli.command('list', help='List the Plans')
@pass_context
def list(ctx):
    plans = Plan.list_all(ctx.paths())
    table = utils.table('Plans', plans)
    click.echo(table)
```

4.6 API - Application Programming Interface

QTIP consists of different tools(metrics) to benchmark the NFVI. These metrics fall under different NFVI subsystems(QPI's) such as compute, storage and network. A plan consists of one or more QPI's, depending upon how the end-user would want to measure performance. API is designed to expose a RESTful interface to the user for executing benchmarks and viewing respective scores.

4.6.1 Framework

QTIP API has been created using the Python package Connexion. It has been chosen for a number of reasons. It follows API First approach to create micro-services. Hence, firstly the API specifications are defined from the client side perspective, followed by the implementation of the micro-service. It decouples the business logic from routing and resource mapping making design and implementation cleaner.

It has two major components:

API Specifications

> The API specification is defined in a yaml or json file. Connexion follows Open API specification to determine the design and maps the endpoints to methods in python.

Micro-service Implementation Connexion maps the `operationId` corresponding to every operation in API Specification to methods in python which handles request and responses.

As explained, QTIP consists of metrics, QPI's and plans. The API is designed to provide a RESTful interface to all these components. It is responsible to provide listing and details of each individual element making up these components.

4.6.2 Design

Specification

API's entry point (`main`) runs connexion `App` class object after adding API Specification using `App.add_api` method. It loads specification from `swagger.yaml` file by specifying `specification_dir`.

Connexion reads API's endpoints(paths), operations, their request and response parameter details and response definitions from the API specification i.e. `swagger.yaml` in this case.

Following example demonstrates specification for the resource `plans`.

```
paths:
  /plans/{name}:
    get:
```

(continues on next page)

(continued from previous page)

```
      summary: Get a plan by plan name
      operationId: qtip.api.controllers.plan.get_plan
      tags:
        - Plan
        - Standalone
      parameters:
        - name: name
          in: path
          description: Plan name
          required: true
          type: string
      responses:
        200:
          description: Plan information
          schema:
            $ref: '#/definitions/Plan'
        404:
          description: Plan not found
          schema:
            $ref: '#/definitions/Error'
        501:
          description: Resource not implemented
          schema:
            $ref: '#/definitions/Error'
        default:
          description: Unexpected error
          schema:
            $ref: '#/definitions/Error'
definitions:
  Plan:
    type: object
    required:
      - name
    properties:
      name:
        type: string
      description:
        type: string
      info:
        type: object
      config:
        type: object
```

Every `operationId` in above operations corresponds to a method in controllers. QTIP has three controller modules each for plan, QPI and metric. Connexion will read these mappings and automatically route endpoints to business logic.

Swagger Editor can be explored to play with more such examples and to validate the specification.

Controllers

The request is handled through these methods and response is sent back to the client. Connexion takes care of data validation.

```
@common.check_endpoint_for_error(resource='Plan')
def get_plan(name):
```

(continues on next page)

(continued from previous page)

```
    plan_spec = plan.Plan(name)
    return plan_spec.content
```

In above code `get_plan` takes a plan name and return its content.

The decorator `check_endpoint_for_error` defined in `common` is used to handle error and return a suitable error response.

During Development the server can be run by passing specification file(`swagger.yaml` in this case) to connexion cli -

```
connexion run <path_to_specification_file> -v
```

4.6.3 Extending the Framework

Modifying Existing API:

API can be modified by adding entries in `swagger.yaml` and adding the corresponding controller mapped from `operationID`.

Adding endpoints:

New endpoints can be defined in `paths` section in `swagger.yaml`. To add a new resource *dummy* -

```
paths:
  /dummies:
    get:
      summary: Get all dummies
      operationId: qtip.api.controllers.dummy.get_dummies
      tags:
        - dummy
      responses:
        200:
          description: Foo information
          schema:
            $ref: '#/definitions/Dummy
        default:
          description: Unexpected error
          schema:
            $ref: '#/definitions/Error'
```

And then model of the resource can be defined in the `definitions` section.

```
definitions:
  Dummy:
    type: object
    required:
      - name
    properties:
      name:
        type: string
      description:
        type: string
      id:
        type: string
```

Adding controller methods: Methods for handling requests and responses for every operation for the endpoint added can be implemented in `controller`.

In `controllers.dummy`

```
def get_dummies():
    all_dummies = [<code to get all dummies>]
    return all_dummies, httplib.OK
```

Adding error responses Decorators for handling errors are defined in `common.py` in `api`.

```
from qtip.api import common

@common.check_endpoint_for_error(resource='dummy',operation='get')
def get_dummies()
    all_dummies = [<code to get all dummies>]
    return all_dummies
```

Adding new API:

API can easily be extended by adding more APIs to `Connexion.App` class object using `add_api` class method.

In `__main__`

```
def get_app():
app = connexion.App(__name__, specification_dir=swagger_dir)
app.add_api('swagger.yaml', base_path='/v1.0', strict_validation=True)
return app
```

Extending it to add new APIs. The new API should have all endpoints mapped using `operationId`.

```
from qtip.api import __main__
my_app = __main__.get_app()
my_app.add_api('new_api.yaml',base_path'api2',strict_validation=True)
my_app.run(host="0.0.0.0", port=5000)
```

4.7 Compute QPI

The compute QPI gives user an overall score for system compute performace.

4.7.1 Summary

The compute QPI are calibrated a ZTE E9000 server as a baseline with score of 2500 points. Higher scores are better, with double the score indicating double the performance. The compute QPI provides three different kinds of scores:

- Workload Scores
- Section Scores
- Compute QPI Scores

4.7.2 Baseline

ZTE E9000 server with an 2 Deca core Intel Xeon CPU processor,128560.0MB Memory.

4.7.3 Workload Scores

Each time a workload is executed QTIP calculates a score based on the computer's performance compared to the baseline performance.

4.7.4 Section Scores

QTIP uses a number of different tests, or workloads, to measure performance. The workloads are divided into five different sections:

Section	Detail	Indication
Arithmetic	Arithmetic workloads measure integer operations floating point operations and mathematical functions with whetstone and dhrystone instructions.	Software with heavy calculation tasks.
Memory	Memory workloads measure memory transfer performance with RamSpeed test.	Software working with large scale data operation.
DPI	DPI workloads measure deep-packet inspection speed by performing nDPI test.	Software working with network packet analysis relies on DPI performance.
SSL	SSL Performance workloads measure cipher speeds by using the OpenSSL tool.	Software working with cipher large amounts data relies on SSL Performance.

A section score is the geometric mean of all the workload scores for workloads that are part of the section. These scores are useful for determining the performance of the computer in a particular area.

4.7.5 Compute QPI Scores

The compute QPI score is the weighted arithmetic mean of the five section scores. The compute QPI score provides a way to quickly compare performance across different computers and different platforms without getting bogged down in details.

4.8 Storage QPI

The storage QPI gives user an overall score for storage performance.

The measurement is done by StorPerf.

4.8.1 System Information

System Information are environmental parameters and factors may affect storage performance:

System Factors	Detail	Extraction Method
Ceph Node List	List of nodes which has ceph-osd roles. For example [node-2, node-3, node-4].	Getting from return result of installer node list CLI command.
Ceph Client RDB Cache Mode	Values: "None", "write-through", "write-back".	Getting from value of "rbd cache" and "rbd cache max dirty" keys in client section of ceph configuration; To enable write-through mode, set rbd cache max dirty to 0.
Ceph Client RDB Cache Size	The RBD cache size in bytes. Default is 32 MiB.	Getting from value of "rdb cache size" key in client section of ceph configuration.
Ceph OSD Tier Cache Mode	Values: "None", "Write-back", "Readonly".	Getting from ceph CLI "ceph report" output info.
Use SSD Backed OSD Cache	Values: "Yes", "No".	Getting from POD description and CEPH CLI "ceph-disk list" output info.
Use SSD For Journal	Values: "Yes", "No".	Getting from POD description and CEPH CLI "ceph-disk list" output info.
Ceph Cluster Network Bandwidth	Values: "1G", "10G", "40G".	Getting from physical interface information in POD description, "ifconfig" output info on ceph osd node, and value of "cluster network" key in global section of ceph configuration.

4.8.2 Test Condition

Test Condition	Detail	Extraction Method
Number of Testing VMs	Number of VMs which are created, during running Storperf test case.	It equals the number of Cinder nodes of the SUT.
Distribution of Testing VMS	Number of VMs on each computer node, for example [(node-2: 1), (node-3: 2))].	Recording the distribution when runing Storperf test case.

4.8.3 Baseline

Baseline is established by testing with a set of work loads:

- Queue depth (1, 2, 8)
- Block size (2KB, 8KB, 16KB)
- Read write - sequential read - sequential write - random read - random write - random mixed read write 70/30

4.8.4 Metrics

- Throughput: data transfer rate
- IOPS: I/O operations per second
- Latency: response time

4.8.5 Workload Scores

For each test run, if an equivalent work load in baseline is available, a score will be calculated by comparing the result to baseline.

4.8.6 Section Scores

Section	Detail	Indication
IOPS	Read write I/O Operation per second under steady state Workloads : random read/write	Important for frequent storage access such as event sinks
Throughput	Read write data transfer rate under steady state Workloads: sequential read/write, block size 16KB	Important for high throughput services such as video server
Latency	Average response latency under steady state Workloads: all	Important for real time applications

Section score is the geometric mean of all workload score.

4.8.7 Storage QPI

Storage QPI is the weighted arithmetic mean of all section scores.

CHAPTER 5

Proposals

5.1 Dashboard

The dashboard gives user an intuitive view of benchmark result.

5.1.1 Purpose

The basic element to be displayed is QPI a.k.a. QTIP Performance Index. But it is also important to show user

1. How is the final score calculated?
2. Under what condition is the test plan executed?
3. How many runs of a performance tests have been executed and is there any deviation?
4. Comparison of benchmark result from different PODs or configuration

5.1.2 Templates

Different board templates are created to satisfy the above requirements.

Composition

QTIP gives a simple score but there must be a complex formula behind it. This view explains the composition of the QPI.

Condition

The condition of a benchmark result includes

- System Under Test

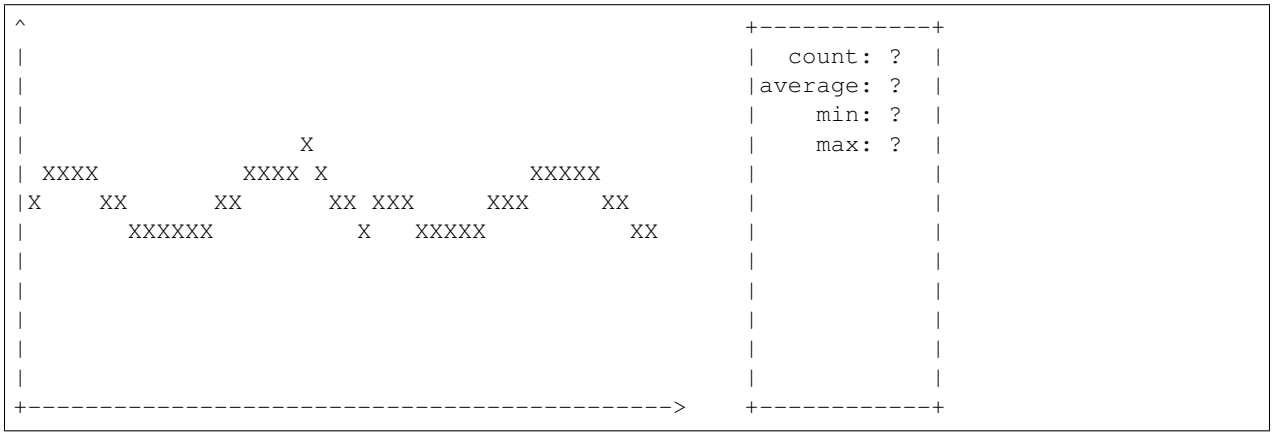
  - Hardware environment
  - Hypervisor version
  - Operation System release version
  - System Configuration
- Test Tools
  - Release version
  - Configuration
- Test Facility
  - Laboratory
  - Engineer
  - Date

Conditions that do NOT have an obvious affect on the test result may be ignored, e.g. temperature, power supply.

Stats

Performance tests are actually measurement of specific metrics. All measurement comes with uncertainty. The final result is normally one or a group of metrics calculated from many repeats.

For each metric, the stats board shall consist of a diagram of all measured values and a box of stats:



The type of diagram and selection of stats shall depend on what metric to show.

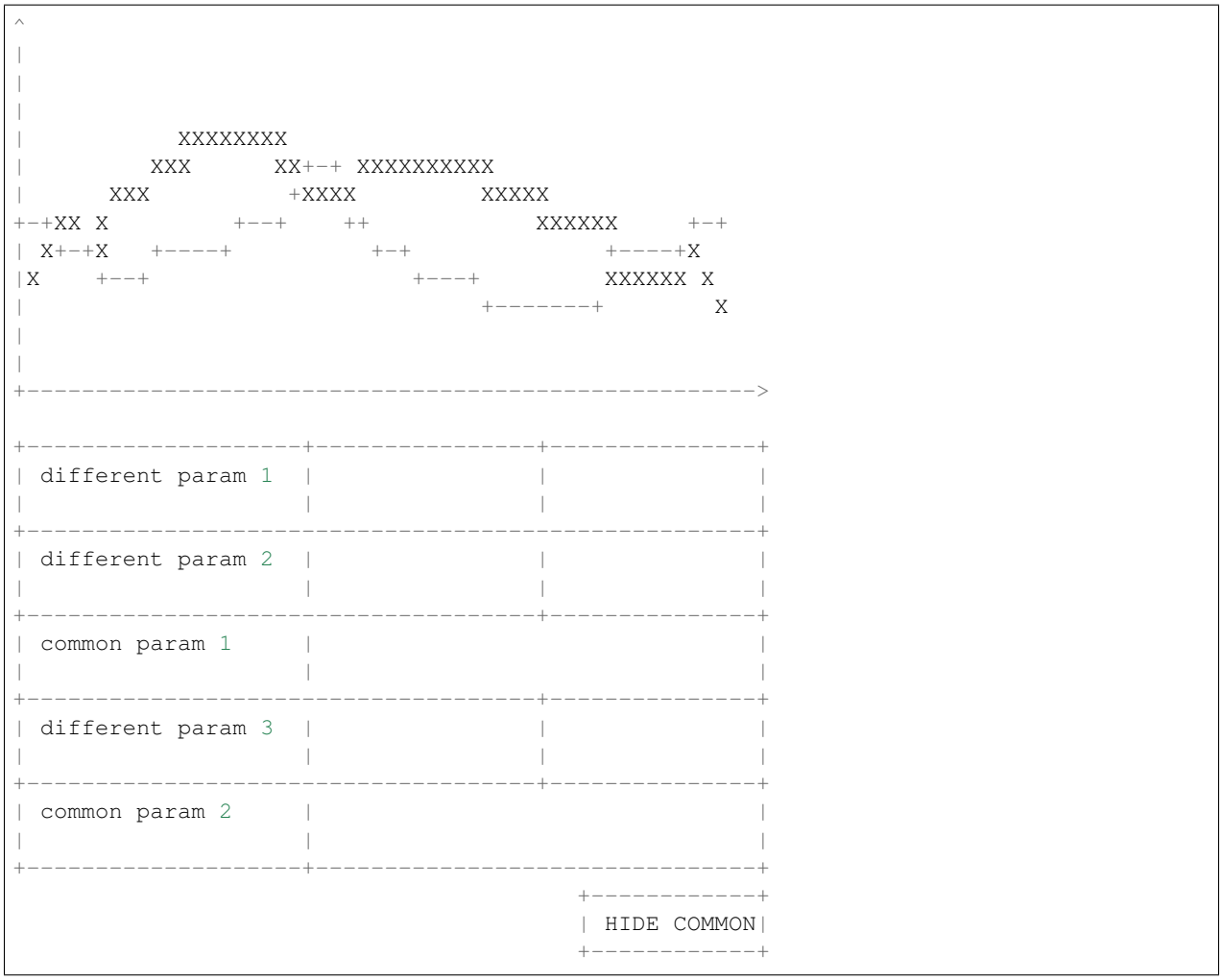
Comparison

Comparison can be done between different PODs or different configuration on the same PODs.

In a comparison view, the metrics are displayed in the same diagram. And the parameters are listed side by side.

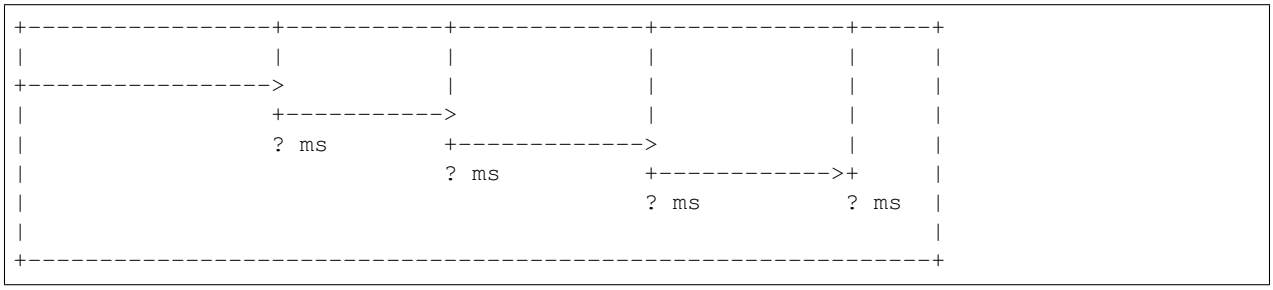
Both common parameters and different parameters are listed. Common values are merged to the same cell. And user may configure the view to hide common rows.

A draft design is as following:



Time line

Time line diagram for analysis of time critical performance test:



The time cost between checkpoints shall be displayed in the diagram.

5.2 Integration with Yardstick

5.2.1 Problem description

For each specified QPI[1], QTIP needs to select a suite of test cases and collect required test results. Based on these results, QTIP calculates the score.

5.2.2 Proposed change

QTIP has a flexible architecture[2] to support different mode: standalone and agent. It is recommended to use **agent mode** to work with existing test runners. Yardstick will act as a runner to generate test result and trigger QTIP agent on the completion of test.

Work Items in Yardstick

1. Create a customized suite in Yardstick

Yardstick not only has many existing suites but also support customized suites. QTIP could create a suite named **QTIP-PoC** in Yardstick repo to verify workflow of QTIP agent mode.

2. Launch QTIP in Yardstick

Whether to launch QTIP will be determined by checking the existence of OS environment variable *QTIP*. If it exists, QTIP will be launched by using Yardstick CLI *yardstick plugin install*[3].

3. Yardstick interacts with QTIP

See Yardstick-QTIP+integration for details.

Work Items in QTIP

1. Provide an API for Yardstick to post test result and environment info

After completing test execution, Yardstick will post test result and enviroment info with JSON format via QTIP API. See Yardstick-QTIP+integration for details.

2. Parse yardstick test result

When QTIP agent receive Yarstick test result and enviroment info, QTIP agent will extract metrics which is definded in metric spec configuration file. Based on these metrics, QTIP agent will caculate QPI.

3. Provide an API for querying QPI

QTIP will provide an API for querying QPI. See Yardstick-QTIP+integration for details.

5.2.3 Implementation

Assignee(s)

Primary assignee: wu.zhihui

Other contributors TBD

[1] QTIP performance index
[2] https://wiki.opnfv.org/display/qtip/Architecture
[3] https://wiki.opnfv.org/display/yardstick/How+to+install+a+plug-in+into+Yardstick

5.2.4 Testing

The changes will be covered by new unit test.

5.2.5 Documentation

TBD

5.2.6 Reference

5.3 Network Performance Indicator

Sridhar K. N. Rao, Spirent Communications

Network performance is an important measure that should be considered for design and deployment of virtual network functions in the cloud. In this document, we propose an indicator for network performance. We consider following parameters for the indicator.

1. The network throughput.
2. The network delay
3. Application SLAs
4. The topology - Path Length and Number of Virtual Network-Elements.
5. Network Virtualization - Vxlan, GRE, VLAN, etc.

The most commonly used, and well measured, network-performance metrics are throughput and delay. However, considering the NFV environments, we add additional metrics to come up with a single indicator value. With these additional metrics, we plan to cover various deployment scenarios of the virtualized network functions.

The proposed network performance indicator value ranges from 0 - 1.0

As majority of indicators, these values should mainly be used for comparative analysis, and not to be seen as a absolute indicator.

Note: Additional parameters such as - total load on the network - can be considered in future.

The network performance indicator (I) can be represented as:

$I = w_t(1 - \frac{E_t - O_t}{E_t}) + w_d(1 - \frac{O_d - E_d}{O_d}) + w_a(1 - \frac{E_a - O_a}{E_a}) + w_s(1 - \frac{T_n - V_n}{T_n}) + w_p(1 - \frac{1}{T_n + 1}) + w_v * C_{nv}$

Where,

Notation	Description	Example Value
w_t	Weightage for the Throughput	0.3
w_d	Weightage for the Delay	0.3
w_a	Weightage for the Application SLA	0.1
w_s	Weightage for the Topology - Network Elements	0.1
w_p	Weightage for the Topology - Path Length	0.1
w_v	Weightage for the Virtualization	0.1

And

Notation	Description
E_t & O_t	Expected (theoretical Max) and Obtained Average Throughput
E_d & O_d	Expected and Otained Minimum Delay
E_a & O_a	Expected and Obtained Application SLA Metric
T_n	Total number of Network Elements (Switches and Routers)
V_n	Total number of Virtual Network Elements
C_{nv}	Network Virtualization Constant

It would be interesting to explore the following alternative:

$I = I_E - I_O$

where

$I_E = w_t * E_t + w_d * \frac{1}{E_d} + w_a . \frac{1}{E_a} + w_s * \frac{1}{T_n} + w_p * V_n + W_v * C_{nv}$

and

$I_O = w_t * O_t + w_d * \frac{1}{O_d} + w_a . \frac{1}{O_a} + w_s * \frac{1}{T_n} + w_p * V_n + W_v * C_{nv}$