vmtp Documentation

Release 2.4.5

OpenStack Foundation

Jan 19, 2018

Contents

Contents:

CHAPTER 1

Overview

VMTP is a data path performance measurement tool for OpenStack clouds.

1.1 Features

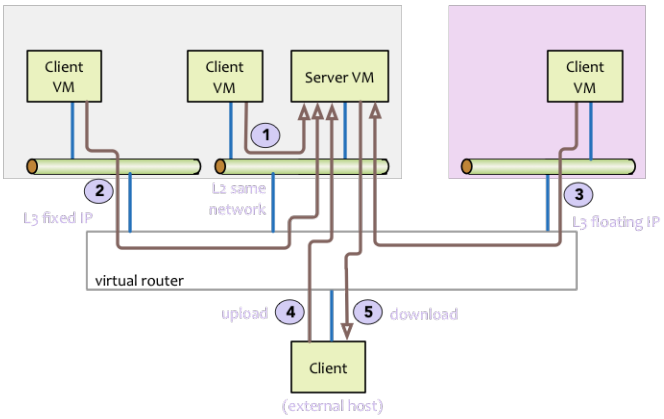
Have you ever had the need for a quick, simple and automatable way to get VM-level or host-level single-flow throughput and latency numbers from any OpenStack cloud, and take into account various Neutron topologies? Or check whether some OpenStack configuration option, Neutron plug-in performs to expectation or if there is any data path impact for upgrading to a different OpenStack release?

VMTP is a small python application that will automatically perform ping connectivity, round trip time measurement (latency) and TCP/UDP throughput measurement for the following East/West flows on any OpenStack deployment:

- VM to VM same network (private fixed IP, flow #1)
- VM to VM different network using fixed IP (same as intra-tenant L3 fixed IP, flow #2)
- VM to VM different network using floating IP and NAT (same as floating IP inter-tenant L3, flow #3)

Optionally, when an external Linux host is available for testing North/South flows:

- External host/VM download and upload throughput/latency (L3/floating IP, flow #4 and #5)



Optionally, when SSH login to any Linux host (native or virtual) is available:

- Host to host process-level throughput/latency (intra-node and inter-node)

Optionally, VMTP can extract automatically CPU usage from all native hosts in the cloud during the throughput tests, provided the Ganglia monitoring service (gmond) is installed and enabled on those hosts.

For VM-related flows, VMTP will automatically create the necessary OpenStack resources (router, networks, subnets, key pairs, security groups, test VMs) using the public OpenStack API, install the test tools then orchestrate them to gather the throughput measurements then cleanup all related resources before exiting.

See the usage page for the description of all the command line arguments supported by VMTP.

1.2 Online Documentation

The complete documentation for VMTP including installation and usage guide is available at http://vmtp.readthedocs.io/en/latest

1.3 Pre-requisite

VMTP runs on any Python 2.X envirnment (validated on Linux and MacOSX).

1.3.1 For VM related performance measurements

- Access to the cloud Horizon Dashboard (to retrieve the openrc file)
- 1 working external network pre-configured on the cloud (VMTP will pick the first one found)
- At least 2 floating IP if an external router is configured or 3 floating IP if there is no external router configured
- 1 Linux image available in OpenStack (any distribution)
- A configuration file that is properly set for the cloud to test (see “Configuration File” section below)

1.3.2 For native/external host throughputs

- A public key must be installed on the target hosts (see ssh password-less access below)

1.3.3 For pre-existing native host throughputs

- Firewalls must be configured to allow TCP/UDP ports 5001 and TCP port 5002

1.4 Sample Results Output

VMTP will display the results to stdout with the following data:

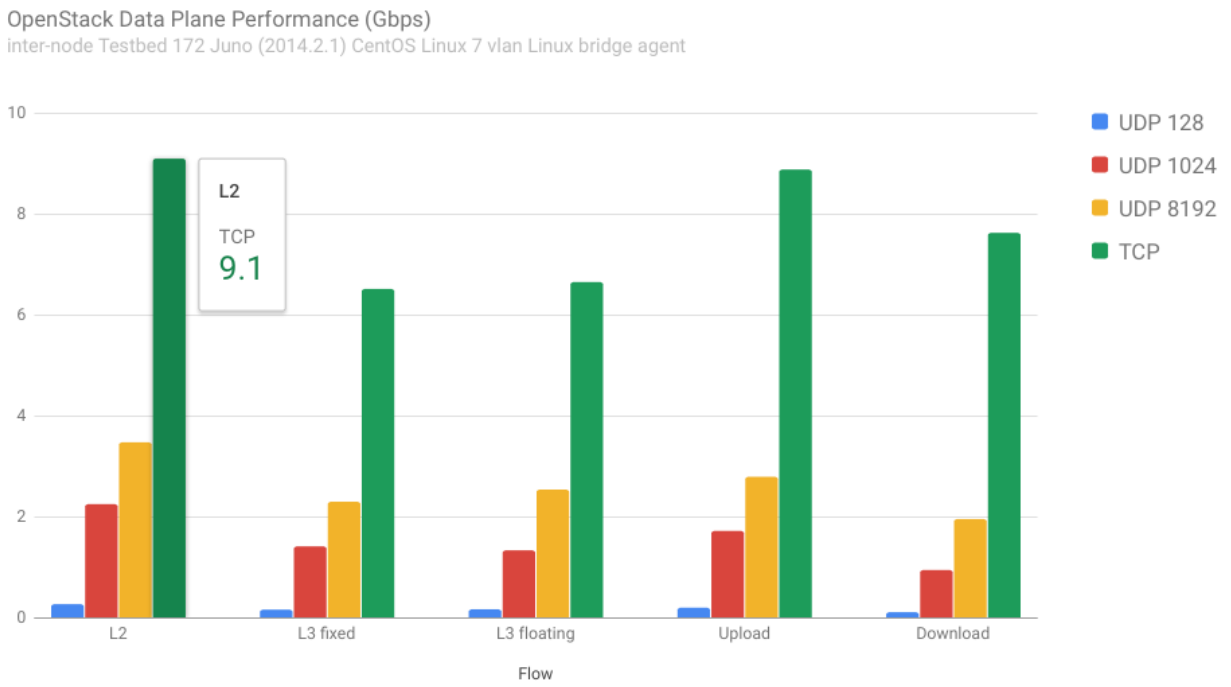
```
Summary of results
==================
Total Scenarios:   22
Passed Scenarios:  17 [100.00%]
Failed Scenarios:  0 [0.00%]
Skipped Scenarios: 5
+----------+-----------------------------------------------------+--------------------+---
↪-------------------------------------------------------------------------------+
| Scenario | Scenario Name                                       | Functional Status |␣
↪Data                                                                           |
+----------+-----------------------------------------------------+--------------------+---
↪-------------------------------------------------------------------------------+
| 1.1      | Same Network, Fixed IP, Intra-node, TCP             | PASSED             | {
↪'tp_kbps': '19262752', 'rtt_ms': '0.38'}                                       |
| 1.2      | Same Network, Fixed IP, Intra-node, UDP             | PASSED             |
↪{128: {'tp_kbps': 243360, 'loss_rate': 0.0}, 1024: {'tp_kbps': 1790414,        |
|          |                                                     |                    |
↪'loss_rate': 0.0}, 8192: {'tp_kbps': 9599648, 'loss_rate': 0.0}}               |
| 1.3      | Same Network, Fixed IP, Intra-node, ICMP            | PASSED             | {
↪'rtt_avg_ms': '0.385', 'rtt_min_ms': '0.237', 'rtt_max_ms': '0.688',           |
|          |                                                     |                    |
↪'rtt_stddev': '0.156'}                                                         |
| 2.1      | Same Network, Fixed IP, Inter-node, TCP             | PASSED             | {
↪'tp_kbps': '5987943', 'rtt_ms': '0.49'}                                        |
| 2.2      | Same Network, Fixed IP, Inter-node, UDP             | PASSED             |
↪{128: {'tp_kbps': 240518, 'loss_rate': 0.0}, 1024: {'tp_kbps': 1804851,        |
|          |                                                     |                    |
↪'loss_rate': 0.0}, 8192: {'tp_kbps': 3074557, 'loss_rate': 0.04}}              |
| 2.3      | Same Network, Fixed IP, Inter-node, ICMP            | PASSED             | {
↪'rtt_avg_ms': '0.601', 'rtt_min_ms': '0.507', 'rtt_max_ms': '0.846',           |
|          |                                                     |                    |
↪'rtt_stddev': '0.126'}                                                         |
| 3.1      | Different Network, Fixed IP, Intra-node, TCP        | PASSED             | {
↪'tp_kbps': '7308597', 'rtt_ms': '0.68'}                                        |
| 3.2      | Different Network, Fixed IP, Intra-node, UDP        | PASSED             |
↪{128: {'tp_kbps': 194764, 'loss_rate': 4.88}, 1024: {'tp_kbps': 1587951,       |
|          |                                                     |                    |
↪'loss_rate': 3.39}, 8192: {'tp_kbps': 2666969, 'loss_rate': 0.0}}              |
| 3.3      | Different Network, Fixed IP, Intra-node, ICMP       | PASSED             | {
↪'rtt_avg_ms': '0.689', 'rtt_min_ms': '0.638', 'rtt_max_ms': '0.761',           |
|          |                                                     |                    |
↪'rtt_stddev': '0.053'}                                                         |
| 4.1      | Different Network, Fixed IP, Inter-node, TCP        | PASSED             | {
↪'tp_kbps': '8487326', 'rtt_ms': '0.713333'}                                    |
| 4.2      | Different Network, Fixed IP, Inter-node, UDP        | PASSED             |
↪{128: {'tp_kbps': 200641, 'loss_rate': 0.0}, 1024: {'tp_kbps': 1198920,        |
|          |                                                     |                    |
↪'loss_rate': 30.54}, 8192: {'tp_kbps': 2657355, 'loss_rate': 0.0}}             |
```

```
| 4.3       | Different Network, Fixed IP, Inter-node, ICMP     | PASSED              | {
↪'rtt_avg_ms': '0.710', 'rtt_min_ms': '0.674', 'rtt_max_ms': '0.729',              |
|           |                                                   |                     |
↪'rtt_stddev': '0.025'}                                                             |
| 5.1       | Different Network, Floating IP, Intra-node, TCP   | PASSED              | {
↪'tp_kbps': '7462958', 'rtt_ms': '0.676667'}                                        |
| 5.2       | Different Network, Floating IP, Intra-node, UDP   | PASSED              |
↪{128: {'tp_kbps': 188808, 'loss_rate': 2.34}, 1024: {'tp_kbps': 1513660,           |
|           |                                                   |                     |
↪'loss_rate': 0.0}, 8192: {'tp_kbps': 2586232, 'loss_rate': 0.0}}                   |
| 5.3       | Different Network, Floating IP, Intra-node, ICMP  | PASSED              | {
↪'rtt_avg_ms': '0.592', 'rtt_min_ms': '0.477', 'rtt_max_ms': '0.663',              |
|           |                                                   |                     |
↪'rtt_stddev': '0.065'}                                                             |
| 6.1       | Different Network, Floating IP, Inter-node, TCP   | PASSED              | {
↪'tp_kbps': '8486828', 'rtt_ms': '0.663333'}                                        |
| 6.2       | Different Network, Floating IP, Inter-node, UDP   | PASSED              |
↪{128: {'tp_kbps': 190434, 'loss_rate': 0.12}, 1024: {'tp_kbps': 1518300,           |
|           |                                                   |                     |
↪'loss_rate': 0.0}, 8192: {'tp_kbps': 2569370, 'loss_rate': 0.0}}                   |
| 6.3       | Different Network, Floating IP, Inter-node, ICMP  | PASSED              | {
↪'rtt_avg_ms': '0.674', 'rtt_min_ms': '0.657', 'rtt_max_ms': '0.702',              |
|           |                                                   |                     |
↪'rtt_stddev': '0.015'}                                                             |
| 7.1       | Native Throughput, TCP                            | SKIPPED             | {}
↪                                                                                   |
| 7.2       | Native Throughput, UDP                            | SKIPPED             | {}
↪                                                                                   |
| 7.3       | Native Throughput, ICMP                           | SKIPPED             | {}
↪                                                                                   |
| 8.1       | VM to Host Uploading                              | SKIPPED             | {}
↪                                                                                   |
| 8.2       | VM to Host Downloading                            | SKIPPED             | {}
↪                                                                                   |
+-----------+---------------------------------------------------+---------------------+---
↪-------------------------------------------------------------------------------+
```

Detailed results can also be stored in a file in JSON format using the *–json* command line argument and/or stored directly into a MongoDB server. See example.json for an example JSON file that is generated by VMTP.

The packaged python tool genchart.py can be used to generate from the JSON result files column charts in HTML format visible from any browser.

Example of column chart generated by genchart.py:



Property	Value
NIC name	Cisco Systems Inc VIC Ethernet NIC (rev a2)
CPU Info	32 * Intel(R) Xeon(R) CPU E5-2665 0 @ 2.40GHz
VMTP version	2.0.4
OpenStack release	Juno (2014.2.1)
Description	Testbed 172
Date	2015-04-27 12:06:05
Encapsulation	vlan
L2 agent type	Linux bridge agent
Host Linux distribution	CentOS Linux 7

1.5 Limitations and Caveats

VMTP only measures performance for single-flows at the socket/TCP/UDP level (in a VM or natively). Measured numbers therefore reflect what most applications will see.

It is not designed to measure driver level data path performance from inside a VM (such as bypassing the kernel TCP stack and write directly to virtio), there are better tools that can address this type of mesurement.

VMTP ships with pre-built binaries that will run on most x86_64 Linux VMs (which is the vast majority of copute nodes) - see Licensing. Running VMTP on compute nodes that have a different CPU architecture will require rebuilding these binaries for the proper target.

1.6 Licensing

VMTP is licensed under Apache License 2.0 and comes packaged with the following Linux x86_64 binaries for convenience:

- iperf 2.0.5: BSD License (https://iperf.fr/license.html, built from source code: https://sourceforge.net/projects/iperf/files/iperf-2.0.5.tar.gz/download)

- nuttcp: GPL v2 License (http://nuttcp.net/nuttcp/beta/LICENSE, built from source code: http://nuttcp.net/nuttcp/beta/nuttcp-7.3.2.c)

Redistribution of nuttcp and iperf is governed by their respective licenses. Please make sure you read and understand each one before further redistributing VMTP downstream.

1.6.1 Required legal attachment for iperf binary distribution

iperf 2.0.5: built from source code: https://sourceforge.net/projects/iperf/files/iperf-2.0.5.tar.gz/download

(extract from the COPYING file as required by the iperf license, the full copy of the LICENSE is provided under legal/iperf) Copyright (c) 1999-2007, The Board of Trustees of the University of Illinois All Rights Reserved.

Iperf performance test Mark Gates Ajay Tirumala Jim Ferguson Jon Dugan Feng Qin Kevin Gibbs John Estabrook National Laboratory for Applied Network Research National Center for Supercomputing Applications University of Illinois at Urbana-Champaign http://www.ncsa.uiuc.edu

Permission is hereby granted, free of charge, to any person obtaining a copy of this software (Iperf) and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimers.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimers in the documentation and/or other materials provided with the distribution.

Neither the names of the University of Illinois, NCSA, nor the names of its contributors may be used to endorse or promote products derived from this Software without specific prior written permission. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE CONTIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.6.2 Required legal attachment for nuttcp binary distribution

nuttcp 7.3.2c: GPL v2 License (http://nuttcp.net/nuttcp/beta/LICENSE, built from unmodified source code: http://nuttcp.net/nuttcp/beta/nuttcp-7.3.2.c) A copy of the LICENSE file and source code (unmodifed) is provided in this repository (under legal/nuttcp), as required by the nuttcp license.

Extract of interest related to the binary attachment:

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

2. (Provision does not apply since the code is unmodified)

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

> a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

1.7 Links

- Documentation: http://vmtp.readthedocs.io/en/latest
- Source: http://git.openstack.org/cgit/openstack/vmtp
- Supports/Bugs: https://launchpad.net/vmtp
- Mailing List: vmtp-core@lists.launchpad.net

CHAPTER 2

Installation and Quick Start Guides

VMTP can be installed on any server that has access to the OpenStack API. There are 3 ways to install and run the VMTP tool:

- Docker container
- PyPI package
- Git

The Git installation is required for VMTP developers and can be used for users who want to have access to the VMTP source code. Otherwise, most users should select the Docker container installation (if already using Docker containers) or the more "pythonic" method with PyPI packages/pip.

2.1 VMTP Docker Container Installation and Quick Start Guide

Using the VMTP Docker container is the simplest and fastest method if you are already using Docker.

2.1.1 1. Installation

VMTP is available as a container in Docker Hub at berrypatch/vmtp

To pull the latest vmtp container image (you may not need to run these commands with "sudo" if you have root permission):

```
sudo docker pull berrypatch/vmtp
```

To test vmtp is working:

```
sudo docker run --rm berrypatch/vmtp vmtp -h
```

2.1.2 2. Key Pair

VMTP requires a key pair to use to ssh to the test VMs that it will launch in OpenStack. You can use the current user's key pair (located in $HOME/.ssh on the host where you run the container) if they exist:

```
$ ls -l ~/.ssh/id*
-rw------- 1 localadmin localadmin 1679 Mar  9  2015 /home/localadmin/.ssh/id_rsa
-rw-r--r-- 1 localadmin localadmin  400 Mar  9  2015 /home/localadmin/.ssh/id_rsa.pub
```

Otherwise you need to create a key pair on your host:

```
ssh-keygen -t rsa
```

2.1.3 3. Download RC file

VMTP requires downloading an "openrc" file from the OpenStack Dashboard (Project|Acces&Security!Api Access|Download OpenStack RC File). That RC file is required to connect to OpenStack using the OpenStack API. This file should be passed to VMTP using the *-r* option or should be sourced prior to invoking VMTP. Because VMTP runs in a Docker container, the RC file must be accessible from the container. The simplest is to download the RC file on the Docker host and make the (host) directory directly accessible to the container using the -v option. For example if the RC file is downloaded as "admin-openrc.sh" in the current directory, that file is accessible from the container by mapping the current host directory ($PWD) to the "/tmp/vmtp" directory in the container and using the "/tmp/vmtp/admin-openrc.sh" pathname (in the below example, the container command just lists the rc file before exiting)

```
sudo docker run --rm -v $PWD:/tmp/vmtp berrypatch/vmtp ls /tmp/vmtp/admin-openrc.sh
```

2.1.4 4. Preparation steps with OpenStack

Now that the RC file is available from the container, you can run any OpenStack CLI command in the container (since the container will have all standard OpenStack client packages installed along with VMTP). Run the container in interactive mode, get to its shell prompt and source the RC file:

```
sudo docker run --rm -it -v $PWD:/tmp/vmtp berrypatch/vmtp bash
root@af7cedc3866f:/# . /tmp/vmtp/admin-openrc.sh
```

4.1. Verify flavor names

If you are planning to reuse an existing flavor, we will have to check the flavor names available to select one flavor that VMTP should use to launch VM instances. List the flavors (results may be different):

```
root@af7cedc3866f:/# nova flavor-list
+----+-----------+-----------+------+-----------+------+-------+-------------+--------
↪---+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor | Is_
↪Public |
+----+-----------+-----------+------+-----------+------+-------+-------------+--------
↪---+
| 1  | m1.tiny   | 512       | 1    | 0         |      | 1     | 1.0         | True  ␣
↪   |
| 2  | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0         | True  ␣
↪   |
```

```
| 3  | m1.medium | 4096      | 40   | 0          |      | 2     | 1.0          | True  ␣
↪   |
| 4  | m1.large  | 8192      | 80   | 0          |      | 4     | 1.0          | True  ␣
↪   |
| 5  | m1.xlarge | 16384     | 160  | 0          |      | 8     | 1.0          | True  ␣
↪   |
+----+-----------+-----------+------+------------+------+-------+--------------+--------
↪---+
root@af7cedc3866f:/# exit
exit
localadmin@GG27-16:~/wsvmtp$
```

4.2. Upload any Linux VM image to OpenStack

VMTP requires a standard Linux VM image to run its tests in OpenStack. You can skip this step if you already have a standard Linux VM image in your OpenStack (Ubuntu, Fedora, RHEL...).

Otherwise, you can upload any Linux VM image using the glance CLI or using the Horizon dashboard. In the example below we will upload the Ubuntu 14.04 cloud image available from the uec-images.ubuntu.com web site using the glance CLI and we will name it "Ubuntu Server 14.04".

If your OpenStack can access directly the Internet:

```
root@af7cedc3866f:/# glance --os-image-api-version 1 image-create --copy-from http://
↪uec-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-uefi1.img --disk-
↪format qcow2 --container-format bare --name 'Ubuntu Server 14.04'
```

The glance command will return immediately but it will take some time for the file to get transferred. You will need to check for the status of the image before you can use it (will "queued", then "saving" then "active" if there is no issue).

If you prefer to make a local copy of the image, from another terminal window on the host:

```
wget http://uec-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-uefi1.
↪img
```

Then copy it to OpenStack using the glance CLI (from the container prompt):

```
root@af7cedc3866f:/# glance --os-image-api-version 1 image-create --file /tmp/vmtp/
↪trusty-server-cloudimg-amd64-uefi1.img --disk-format qcow2 --container-format bare -
↪-name 'Ubuntu 14.04'
```

Then list the images to verify and exit the container:

```
root@af7cedc3866f:/# glance image-list
+--------------------------------------+---------------------+
| ID                                   | Name                |
+--------------------------------------+---------------------+
| 5d7899d9-811c-483f-82b3-282a9bf143bf | cirros              |
| 443ee290-b714-4bfe-9acb-b996ed6cc118 | Ubuntu 14.04        |
+--------------------------------------+---------------------+
root@af7cedc3866f:/# glance image-show 443ee290-b714-4bfe-9acb-b996ed6cc118
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | 479a314d90cefc163fdcfb875a070cd8     |
| container_format | bare                                 |
| created_at       | 2016-07-04T17:53:20Z                 |
```

```
| disk_format      | qcow2                                |
| id               | 443ee290-b714-4bfe-9acb-b996ed6cc118 |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
| name             | Ubuntu 14.04                         |
| owner            | 5d912149f7474804824a463464874a21     |
| protected        | False                                |
| size             | 268829184                            |
| status           | active                               |
| tags             | []                                   |
| updated_at       | 2016-07-04T18:06:38Z                 |
| virtual_size     | None                                 |
| visibility       | private                              |
+------------------+--------------------------------------+
root@af7cedc3866f:/# exit
```

2.1.5 5. Create your VMTP config file

Get a copy of the default VMTP configuration file and save it in the local directory:

```
sudo docker run --rm -v $PWD:/tmp/vmtp berrypatch/vmtp vmtp -sc > vmtp.cfg
```

Edit the vmtp.cfg file and make sure the following parameters are set properly:

- "image_name" must be the image name to use by VMTP ('Ubuntu Server 14.04' in the above example)
- "ssh_vm_username" must be a valid user name for the Linux image ("ubuntu" for Ubuntu images)
- "flavor_type" must be either an appropriate flavor name (step 4.1 above) or a custom flavor will be created with "flavor_type" name and specification declared in "flavor" config.
- "flavor" must be the specification of a custom flavor that will be created in case "flavor_type" is non-existing in OpenStack.
- "public_key_file" must point to your public key (see below)
- "private_key_file" must point to your private key (see below)

To access the key pairs from the container, the simplest is to map the $HOME/.ssh directory to /tmp/ssh in the container (for example):

```
sudo docker run --rm -v $HOME/.ssh:/tmp/ssh berrypatch/vmtp ls -l /tmp/ssh/id_rsa
```

With this mapping, the key pair parameters in the config file should be set to:

```
public_key_file: /tmp/ssh/id_rsa.pub
private_key_file: /tmp/ssh/id_rsa
```

2.1.6 6. Run VMTP

Docker options used:

- –rm : remove the container instance after execution
- -it : interactive mode + use a terminal
- -v $PWD:/tmp/vmtp : map the host current directory to /tmp/vmtp in the container

- -v $HOME/.ssh:/tmp/ssh : map the host $HOME/.ssh directory to /tmp/ssh in the container

VMTP options used:

- -d : debug mode (more verbose)
- -c /tmp/vmtp/vmtp.cfg : specify the config file to use
- -r /tmp/vmtp/admin-openrc.sh : specify the RC file to use
- -p secret : specify the OpenStack password to use (replace with your own password)
- –protocol T : only do TCP throughput test (shorter time)
- –json /tmp/vmtp/test.json : save results in json format to a file

```
sudo docker run --rm -it -v $PWD:/tmp/vmtp -v $HOME/.ssh:/tmp/ssh berrypatch/vmtp␣
↪vmtp -d -c /tmp/vmtp/vmtp.cfg -r /tmp/vmtp/admin-openrc.sh -p secret --protocol T --
↪json /tmp/vmtp/test.json
```

This should produce an output similar to this (a complete run with the above options should take around 15 minutes but may vary based on the control plane speed of your OpenStack cloud):

```
Using http://172.29.86.28:5000/v2.0
VM public key:  /tmp/ssh/id_rsa.pub
VM private key: /tmp/ssh/id_rsa
Found image Ubuntu Server 14.04 to launch VM, will continue
Using external network: ext-net
Found external router: demo-router
Created internal network: pns-internal-net
Created internal network: pns-internal-net2
Ext router associated to pns-internal-net
Ext router associated to pns-internal-net2
OpenStack agent: Open vSwitch agent
OpenStack network type: vlan
[TestServer1] Creating server VM...
[TestServer1] Starting on zone nova:compute-server-2
[TestServer1] VM status=BUILD, retrying 1 of 50...
[TestServer1] VM status=BUILD, retrying 2 of 50...
...
[TestServer1] Floating IP 10.23.220.45 created
[TestServer1] Started - associating floating IP 10.23.220.45
[TestServer1] Internal network IP: 192.168.1.3
[TestServer1] SSH IP: 10.23.220.45
[TestServer1] Setup SSH for ubuntu@10.23.220.45
[TestServer1] Installing nuttcp-7.3.2...
[TestServer1] Copying nuttcp-7.3.2 to target...
[TestServer1] Starting nuttcp-7.3.2 server...
[TestServer1]
[TestClient1] Creating client VM...
[TestClient1] Starting on zone nova:compute-server-2
[TestClient1] VM status=BUILD, retrying 1 of 50...
[TestClient1] VM status=BUILD, retrying 2 of 50...
...
[TestClient1] Floating IP 10.23.220.46 created
[TestClient1] Started - associating floating IP 10.23.220.46
[TestClient1] Internal network IP: 192.168.1.4
[TestClient1] SSH IP: 10.23.220.46
[TestClient1] Setup SSH for ubuntu@10.23.220.46
[TestClient1] Installing nuttcp-7.3.2...
[TestClient1] Copying nuttcp-7.3.2 to target...
```

```
============================================================
Flow 1: VM to VM same network fixed IP (intra-node)
[TestClient1] Measuring TCP Throughput (packet size=65536)...
[TestClient1] /tmp/nuttcp-7.3.2 -T10  -l65536 -p5001 -P5002 -fparse 192.168.1.3
[TestClient1] megabytes=20329.1875 real_seconds=10.00 rate_Mbps=17049.6212 tx_cpu=92␣
↪rx_cpu=53 retrans=0 rtt_ms=0.47
...
{   'az_from': u'nova:compute-server-2',
    'az_to': u'nova:compute-server-2',
    'desc': 'VM to VM same network fixed IP (intra-node)',
    'distro_id': 'Ubuntu',
    'distro_version': '14.04',
    'ip_from': u'192.168.1.4',
    'ip_to': u'192.168.1.3',
    'results': [   {   'pkt_size': 65536,
                       'protocol': 'TCP',
                       'rtt_ms': 0.47,
                       'throughput_kbps': 17458812,
                       'tool': 'nuttcp-7.3.2'},
                   {   'pkt_size': 65536,
                       'protocol': 'TCP',
                       'rtt_ms': 0.19,
                       'throughput_kbps': 13832383,
                       'tool': 'nuttcp-7.3.2'},
                   {   'pkt_size': 65536,
                       'protocol': 'TCP',
                       'rtt_ms': 0.21,
                       'throughput_kbps': 17130867,
                       'tool': 'nuttcp-7.3.2'}]}
[TestClient1] Floating IP 10.23.220.46 deleted
[TestClient1] Instance deleted
[TestClient2] Creating client VM...
[TestClient2] Starting on zone nova:compute-server-2
[TestClient2] VM status=BUILD, retrying 1 of 50...
[TestClient2] VM status=BUILD, retrying 2 of 50...

...

---- Cleanup ----
[TestServer1] Terminating nuttcp-7.3.2
[TestServer1] Floating IP 10.23.220.45 deleted
[TestServer1] Instance deleted
Network pns-internal-net deleted
Network pns-internal-net2 deleted
Removed public key pns_public_key
Deleting security group

Summary of results
==================
Total Scenarios:   22
Passed Scenarios:  5 [100.00%]
Failed Scenarios:  0 [0.00%]
Skipped Scenarios: 17
+----------+-----------------------------------------------------+-------------------+---
↪----------------------------------------+
| Scenario | Scenario Name                                       | Functional Status |␣
↪Data                                    |
+----------+-----------------------------------------------------+-------------------+---
↪----------------------------------------+
```

```
| 1.1       | Same Network, Fixed IP, Intra-node, TCP           | PASSED            | {
↪'tp_kbps': '16140687', 'rtt_ms': '0.29'}     |
| 1.2       | Same Network, Fixed IP, Intra-node, UDP           | SKIPPED           | {}
↪                                                 |
| 1.3       | Same Network, Fixed IP, Intra-node, ICMP          | SKIPPED           | {}
↪                                                 |
| 2.1       | Same Network, Fixed IP, Inter-node, TCP           | PASSED            | {
↪'tp_kbps': '4082749', 'rtt_ms': '0.5'}       |
| 2.2       | Same Network, Fixed IP, Inter-node, UDP           | SKIPPED           | {}
↪                                                 |
| 2.3       | Same Network, Fixed IP, Inter-node, ICMP          | SKIPPED           | {}
↪                                                 |
| 3.1       | Different Network, Fixed IP, Intra-node, TCP      | PASSED            | {
↪'tp_kbps': '2371753', 'rtt_ms': '0.386667'} |
| 3.2       | Different Network, Fixed IP, Intra-node, UDP      | SKIPPED           | {}
↪                                                 |
| 3.3       | Different Network, Fixed IP, Intra-node, ICMP     | SKIPPED           | {}
↪                                                 |
| 4.1       | Different Network, Fixed IP, Inter-node, TCP      | PASSED            | {
↪'tp_kbps': '2036303', 'rtt_ms': '0.623333'} |
| 4.2       | Different Network, Fixed IP, Inter-node, UDP      | SKIPPED           | {}
↪                                                 |
| 4.3       | Different Network, Fixed IP, Inter-node, ICMP     | SKIPPED           | {}
↪                                                 |
| 5.1       | Different Network, Floating IP, Intra-node, TCP   | PASSED            | {
↪'tp_kbps': '2260145', 'rtt_ms': '0.476667'} |
| 5.2       | Different Network, Floating IP, Intra-node, UDP   | SKIPPED           | {}
↪                                                 |
| 5.3       | Different Network, Floating IP, Intra-node, ICMP  | SKIPPED           | {}
↪                                                 |
| 6.1       | Different Network, Floating IP, Inter-node, TCP   | PASSED            | {
↪'tp_kbps': '2134303', 'rtt_ms': '0.543333'} |
| 6.2       | Different Network, Floating IP, Inter-node, UDP   | SKIPPED           | {}
↪                                                 |
| 6.3       | Different Network, Floating IP, Inter-node, ICMP  | SKIPPED           | {}
↪                                                 |
| 7.1       | Native Throughput, TCP                            | SKIPPED           | {}
↪                                                 |
| 7.2       | Native Throughput, UDP                            | SKIPPED           | {}
↪                                                 |
| 7.3       | Native Throughput, ICMP                           | SKIPPED           | {}
↪                                                 |
| 8.1       | VM to Host Uploading                              | SKIPPED           | {}
↪                                                 |
| 8.2       | VM to Host Downloading                            | SKIPPED           | {}
↪                                                 |
+-----------+---------------------------------------------------+-------------------+---
↪-----------------------------------------+
Saving results in json file: /tmp/vmtp/test.json...
```

2.1.7 8. Generate the results chart from the JSON result file

Assuming the json result file is saved by the container run the vmtp_genchart container command from the host current directory:

```
$ sudo docker run --rm -v $PWD:/tmp/vmtp berrypatch/vmtp vmtp_genchart -c /tmp/vmtp/
↪test.html /tmp/vmtp/test.json
Generating chart drawing code to /tmp/vmtp/test.html...
$
```

vmtp_genchart options:

- -c /tmp/vmtp/test.html : save the generated html file to the mapped directory
- /tmp/vmtp/test.json : the json file that contains the results of the VMTP run

The fie is available in the current directory and can be viewed with any browser:

```
$ ls -l test.html
-rw-r--r-- 1 root root 1557 Jul  4 14:10 test.html
```

2.2 VMTP PyPI Installation and Quick Start Guide

2.2.1 1. Installation

PyPI (The Python Package Index) is the most popular distribution repository for software in Python. The latest stable version of VMTP can be installed from PyPI using pip.

Step 1

Install required libraries. Run the command based on your distro.

Ubuntu/Debian based:

```
$ sudo apt-get install build-essential python-dev python-pip python-virtualenv git␣
↪git-review
$ sudo apt-get install libxml2-dev libxslt-dev libffi-dev libz-dev libyaml-dev libssl-
↪dev
```

RHEL/CentOS based:

```
$ sudo yum install gcc python-devel python-pip python-virtualenv git
$ sudo yum install libxml2-devel libxslt-devel libffi-devel libyaml-devel openssl-
↪devel
```

MacOSX:

```
$ # Download the XCode command line tools from Apple App Store
$ xcode-select --install
$ sudo easy_install pip
$ sudo pip install virtualenv
```

Step 2

Create a virtual environment for Python, and install VMTP:

```
$ virtualenv ./vmtpenv
$ . ./vmtpenv/bin/activate
$ pip install vmtp
$ vmtp -h
```

Note: "A Virtual Environment is a tool to keep the dependencies required by different projects in separate places, by creating virtual Python environments for them." It is optional but recommended. If isolation among multiple Python projects is not needed, we could use instead:

```
$ sudo pip install vmtp
```

2.2.2 2. Key Pair

VMTP requires a key pair to use to ssh to the test VMs that it will launch in OpenStack. You can use the current user's key pair (located in $HOME/.ssh on the host where you run the container) if they exist:

```
$ ls -l ~/.ssh/id*
-rw------- 1 localadmin localadmin 1679 Mar  9  2015 /home/localadmin/.ssh/id_rsa
-rw-r--r-- 1 localadmin localadmin  400 Mar  9  2015 /home/localadmin/.ssh/id_rsa.pub
```

Otherwise you need to create a key pair on your host:

```
ssh-keygen -t rsa
```

2.2.3 3. Download RC file

VMTP requires downloading an "openrc" file from the OpenStack Dashboard (Project|Acces&Security!Api Access|Download OpenStack RC File). That RC file is required to connect to OpenStack using the OpenStack API. This file should be passed to VMTP using the *-r* option or should be sourced prior to invoking VMTP. In this example we assume the RC file is saved in the current directory under the name "admin-openrc.sh".

2.2.4 4. Preparation steps with OpenStack

In the VMTP virtual environment, you can run any OpenStack CLI command (since the virtual environment will have all standard OpenStack client packages installed along with VMTP). Source the RC file so we can execute the CLI commands:

```
source admin-openrc.sh
```

4.1. Verify flavor names

If you are planning to reuse an existing flavor, we will have to check the flavor names available to select one flavor that VMTP should use to launch VM instances. List the flavors (results may be different):

```
$ nova flavor-list
+----+-----------+-----------+------+-----------+------+-------+-------------+--------
↪---+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor | Is_
↪Public |
```

```
+----+-----------+-----------+------+-----------+------+-------+-------------+--------
↪---+
| 1  | m1.tiny   | 512       | 1    | 0         |      | 1     | 1.0         | True  ␣
↪   |
| 2  | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0         | True  ␣
↪   |
| 3  | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0         | True  ␣
↪   |
| 4  | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0         | True  ␣
↪   |
| 5  | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0         | True  ␣
↪   |
+----+-----------+-----------+------+-----------+------+-------+-------------+--------
↪---+
$
```

Otherwise, a temporary flavor can be created for you from the config file specification (Step 5).

4.2. Upload any Linux VM image to OpenStack

VMTP requires a standard Linux VM image to run its tests in OpenStack. You can skip this step if you already have a standard Linux VM image in your OpenStack (Ubuntu, Fedora, RHEL...).

Otherwise, you can upload any Linux VM image using the glance CLI or using the Horizon dashboard. In the example below we will upload the Ubuntu 14.04 cloud image available from the uec-images.ubuntu.com web site using the glance CLI and we will name it "Ubuntu Server 14.04".

If your OpenStack can access directly the Internet:

```
glance --os-image-api-version 1 image-create --copy-from http://uec-images.ubuntu.com/
↪trusty/current/trusty-server-cloudimg-amd64-uefi1.img --disk-format qcow2 --
↪container-format bare --name 'Ubuntu Server 14.04'
```

The glance command will return immediately but it will take some time for the file to get transferred. You will need to check for the status of the image before you can use it (will "queued", then "saving" then "active" if there is no issue).

If you prefer to make a local copy of the image:

```
wget http://uec-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-uefi1.
↪img
```

Then copy it to OpenStack using the glance CLI:

```
glance --os-image-api-version 1 image-create --file /tmp/vmtp/trusty-server-cloudimg-
↪amd64-uefi1.img --disk-format qcow2 --container-format bare --name 'Ubuntu 14.04'
```

Then list the images to verify:

```
$ glance image-list
+--------------------------------------+---------------------+
| ID                                   | Name                |
+--------------------------------------+---------------------+
| 5d7899d9-811c-483f-82b3-282a9bf143bf | cirros              |
| 443ee290-b714-4bfe-9acb-b996ed6cc118 | Ubuntu 14.04        |
+--------------------------------------+---------------------+
$ glance image-show 443ee290-b714-4bfe-9acb-b996ed6cc118
+------------------+--------------------------------------+
```

```
| Property          | Value                                |
+-------------------+--------------------------------------+
| checksum          | 479a314d90cefc163fdcfb875a070cd8     |
| container_format  | bare                                 |
| created_at        | 2016-07-04T17:53:20Z                 |
| disk_format       | qcow2                                |
| id                | 443ee290-b714-4bfe-9acb-b996ed6cc118 |
| min_disk          | 0                                    |
| min_ram           | 0                                    |
| name              | Ubuntu 14.04                         |
| owner             | 5d912149f7474804824a463464874a21     |
| protected         | False                                |
| size              | 268829184                            |
| status            | active                               |
| tags              | []                                   |
| updated_at        | 2016-07-04T18:06:38Z                 |
| virtual_size      | None                                 |
| visibility        | private                              |
+-------------------+--------------------------------------+
$
```

2.2.5 5. Create your VMTP config file

Get a copy of the default VMTP configuration file and save it in the local directory:

```
vmtp -sc > vmtp.cfg
```

Edit the vmtp.cfg file and make sure the following parameters are set properly:

- "image_name" must be the image name to use by VMTP ('Ubuntu Server 14.04' in the above example)
- "ssh_vm_username" must be a valid user name for the Linux image ("ubuntu" for Ubuntu images)
- "flavor_type" must be either an appropriate flavor name (step 4.1 above) or a custom flavor will be created with "flavor_type" name and specification declared in "flavor" config.
- "flavor" must be the specification of a custom flavor that will be created in case "flavor_type" is non-existing in OpenStack.

2.2.6 6. Run VMTP

VMTP options used:

- -d : debug mode (more verbose)
- -c vmtp.cfg : specify the config file to use
- -r admin-openrc.sh : specify the RC file to use
- -p secret : specify the OpenStack password to use (replace with your own password)
- –protocol T : only do TCP throughput test (shorter time)
- –json test.json : save results in json format to a file

```
vmtp -d -c vmtp.cfg -r admin-openrc.sh -p secret --protocol T --json test.json
```

This should produce an output similar to this (a complete run with the above options should take around 15 minutes but may vary based on the control plane speed of your OpenStack cloud):

```
Using http://172.29.86.28:5000/v2.0
VM public key:  /home/localadmin/.ssh/id_rsa.pub
VM private key: /home/localadmin/.ssh/id_rsa
Found image Ubuntu Server 14.04 to launch VM, will continue
Using external network: ext-net
Found external router: demo-router
Created internal network: pns-internal-net
Created internal network: pns-internal-net2
Ext router associated to pns-internal-net
Ext router associated to pns-internal-net2
OpenStack agent: Open vSwitch agent
OpenStack network type: vlan
[TestServer1] Creating server VM...
[TestServer1] Starting on zone nova:compute-server-2
[TestServer1] VM status=BUILD, retrying 1 of 50...
[TestServer1] VM status=BUILD, retrying 2 of 50...
...
[TestServer1] Floating IP 10.23.220.45 created
[TestServer1] Started - associating floating IP 10.23.220.45
[TestServer1] Internal network IP: 192.168.1.3
[TestServer1] SSH IP: 10.23.220.45
[TestServer1] Setup SSH for ubuntu@10.23.220.45
[TestServer1] Installing nuttcp-7.3.2...
[TestServer1] Copying nuttcp-7.3.2 to target...
[TestServer1] Starting nuttcp-7.3.2 server...
[TestServer1]
[TestClient1] Creating client VM...
[TestClient1] Starting on zone nova:compute-server-2
[TestClient1] VM status=BUILD, retrying 1 of 50...
[TestClient1] VM status=BUILD, retrying 2 of 50...
...
[TestClient1] Floating IP 10.23.220.46 created
[TestClient1] Started - associating floating IP 10.23.220.46
[TestClient1] Internal network IP: 192.168.1.4
[TestClient1] SSH IP: 10.23.220.46
[TestClient1] Setup SSH for ubuntu@10.23.220.46
[TestClient1] Installing nuttcp-7.3.2...
[TestClient1] Copying nuttcp-7.3.2 to target...
============================================================
Flow 1: VM to VM same network fixed IP (intra-node)
[TestClient1] Measuring TCP Throughput (packet size=65536)...
[TestClient1] /tmp/nuttcp-7.3.2 -T10  -l65536 -p5001 -P5002 -fparse 192.168.1.3
[TestClient1] megabytes=20329.1875 real_seconds=10.00 rate_Mbps=17049.6212 tx_cpu=92␣
↪rx_cpu=53 retrans=0 rtt_ms=0.47
...
{   'az_from': u'nova:compute-server-2',
    'az_to': u'nova:compute-server-2',
    'desc': 'VM to VM same network fixed IP (intra-node)',
    'distro_id': 'Ubuntu',
    'distro_version': '14.04',
    'ip_from': u'192.168.1.4',
    'ip_to': u'192.168.1.3',
    'results': [   {   'pkt_size': 65536,
                       'protocol': 'TCP',
                       'rtt_ms': 0.47,
                       'throughput_kbps': 17458812,
```

```
                        'tool': 'nuttcp-7.3.2'},
                    {   'pkt_size': 65536,
                        'protocol': 'TCP',
                        'rtt_ms': 0.19,
                        'throughput_kbps': 13832383,
                        'tool': 'nuttcp-7.3.2'},
                    {   'pkt_size': 65536,
                        'protocol': 'TCP',
                        'rtt_ms': 0.21,
                        'throughput_kbps': 17130867,
                        'tool': 'nuttcp-7.3.2'}]}
[TestClient1] Floating IP 10.23.220.46 deleted
[TestClient1] Instance deleted
[TestClient2] Creating client VM...
[TestClient2] Starting on zone nova:compute-server-2
[TestClient2] VM status=BUILD, retrying 1 of 50...
[TestClient2] VM status=BUILD, retrying 2 of 50...

...

---- Cleanup ----
[TestServer1] Terminating nuttcp-7.3.2
[TestServer1] Floating IP 10.23.220.45 deleted
[TestServer1] Instance deleted
Network pns-internal-net deleted
Network pns-internal-net2 deleted
Removed public key pns_public_key
Deleting security group

Summary of results
==================
Total Scenarios:   22
Passed Scenarios:  5 [100.00%]
Failed Scenarios:  0 [0.00%]
Skipped Scenarios: 17
+----------+-----------------------------------------------+-------------------+---
↪-----------------------------------------+
| Scenario | Scenario Name                                 | Functional Status |␣
↪Data                                    |
+----------+-----------------------------------------------+-------------------+---
↪-----------------------------------------+
| 1.1      | Same Network, Fixed IP, Intra-node, TCP       | PASSED            | {
↪'tp_kbps': '16140687', 'rtt_ms': '0.29'}    |
| 1.2      | Same Network, Fixed IP, Intra-node, UDP       | SKIPPED           | {}
↪                                            |
| 1.3      | Same Network, Fixed IP, Intra-node, ICMP      | SKIPPED           | {}
↪                                            |
| 2.1      | Same Network, Fixed IP, Inter-node, TCP       | PASSED            | {
↪'tp_kbps': '4082749', 'rtt_ms': '0.5'}      |
| 2.2      | Same Network, Fixed IP, Inter-node, UDP       | SKIPPED           | {}
↪                                            |
| 2.3      | Same Network, Fixed IP, Inter-node, ICMP      | SKIPPED           | {}
↪                                            |
| 3.1      | Different Network, Fixed IP, Intra-node, TCP  | PASSED            | {
↪'tp_kbps': '2371753', 'rtt_ms': '0.386667'} |
| 3.2      | Different Network, Fixed IP, Intra-node, UDP  | SKIPPED           | {}
↪                                            |
| 3.3      | Different Network, Fixed IP, Intra-node, ICMP | SKIPPED           | {}
↪                                            |
```

```
| 4.1      | Different Network, Fixed IP, Inter-node, TCP      | PASSED            | {
↪'tp_kbps': '2036303', 'rtt_ms': '0.623333'} |
| 4.2      | Different Network, Fixed IP, Inter-node, UDP      | SKIPPED           | {}
↪                                                |
| 4.3      | Different Network, Fixed IP, Inter-node, ICMP     | SKIPPED           | {}
↪                                                |
| 5.1      | Different Network, Floating IP, Intra-node, TCP   | PASSED            | {
↪'tp_kbps': '2260145', 'rtt_ms': '0.476667'} |
| 5.2      | Different Network, Floating IP, Intra-node, UDP   | SKIPPED           | {}
↪                                                |
| 5.3      | Different Network, Floating IP, Intra-node, ICMP  | SKIPPED           | {}
↪                                                |
| 6.1      | Different Network, Floating IP, Inter-node, TCP   | PASSED            | {
↪'tp_kbps': '2134303', 'rtt_ms': '0.543333'} |
| 6.2      | Different Network, Floating IP, Inter-node, UDP   | SKIPPED           | {}
↪                                                |
| 6.3      | Different Network, Floating IP, Inter-node, ICMP  | SKIPPED           | {}
↪                                                |
| 7.1      | Native Throughput, TCP                            | SKIPPED           | {}
↪                                                |
| 7.2      | Native Throughput, UDP                            | SKIPPED           | {}
↪                                                |
| 7.3      | Native Throughput, ICMP                           | SKIPPED           | {}
↪                                                |
| 8.1      | VM to Host Uploading                              | SKIPPED           | {}
↪                                                |
| 8.2      | VM to Host Downloading                            | SKIPPED           | {}
↪                                                |
+----------+---------------------------------------------------+-------------------+---
↪-----------------------------------------+
Saving results in json file: test.json...
```

2.2.7 8. Generate the results chart from the JSON result file

Assuming the json result file is saved by the container run the vmtp_genchart container command from the host current directory:

```
$ vmtp_genchart -c test.html test.json
Generating chart drawing code to /tmp/vmtp/test.html...
$
```

vmtp_genchart options:

- -c test.html : save the generated html file to the mapped directory
- test.json : the json file that contains the results of the VMTP run

The fie is available in the current directory and can be viewed with any browser:

```
$ ls -l test.html
-rw-r--r-- 1 root root 1557 Jul  4 14:10 test.html
```

2.3 VMTP Git Installation and Quick Start Guide

2.3.1 1. GitHub/OpenStack Repository based Installation

It is recommended to run VMTP inside a virtual environment (this can be skipped if installed in a dedicated VM).

Installation on Ubuntu/Debian

```
$ sudo apt-get install build-essential python-dev python-pip python-virtualenv git␣
↪git-review
$ sudo apt-get install libxml2-dev libxslt-dev libffi-dev libz-dev libyaml-dev libssl-
↪dev
$ # create a virtual environment
$ virtualenv ./vmtpenv
$ . ./vmtpenv/bin/activate
$ git clone git://git.openstack.org/openstack/vmtp
$ cd vmtp
$ pip install -r requirements-dev.txt
$ cd vmtp
$ python vmtp.py -h
```

Installation on RHEL/CentOS

```
$ sudo yum install gcc python-devel python-pip python-virtualenv git
$ sudo yum install libxml2-devel libxslt-devel libffi-devel libyaml-devel openssl-
↪devel
$ sudo pip install git-review
$ # create a virtual environment
$ virtualenv ./vmtpenv
$ . ./vmtpenv/bin/activate
$ git clone git://git.openstack.org/openstack/vmtp
$ cd vmtp
$ pip install -r requirements-dev.txt
$ cd vmtp
$ python vmtp.py -h
```

Installation on MacOSX

VMTP can run natively on MacOSX. These instructions have been verified to work on MacOSX 10.10 (Yosemite).

First, download XCode from App Store, then execute below commands:

```
$ # Download the XCode command line tools
$ xcode-select --install
$ # Install pip
$ sudo easy_install pip
$ # Install python virtualenv
$ sudo pip install virtualenv
$ # create a virtual environment
$ virtualenv ./vmtpenv
$ . ./vmtpenv/bin/activate
$ git clone git://git.openstack.org/openstack/vmtp
$ cd vmtp
```

```
$ pip install -r requirements-dev.txt
$ cd vmtp
$ python vmtp.py -h
```

2.3.2 2. Key Pair

VMTP requires a key pair to use to ssh to the test VMs that it will launch in OpenStack. You can use the current user's key pair (located in $HOME/.ssh on the host where you run the container) if they exist:

```
$ ls -l ~/.ssh/id*
-rw------- 1 localadmin localadmin 1679 Mar  9  2015 /home/localadmin/.ssh/id_rsa
-rw-r--r-- 1 localadmin localadmin  400 Mar  9  2015 /home/localadmin/.ssh/id_rsa.pub
```

Otherwise you need to create a key pair on your host:

```
ssh-keygen -t rsa
```

2.3.3 3. Download RC file

VMTP requires downloading an "openrc" file from the OpenStack Dashboard (Project|Acces&Security!Api Access|Download OpenStack RC File). That RC file is required to connect to OpenStack using the OpenStack API. This file should be passed to VMTP using the *-r* option or should be sourced prior to invoking VMTP. In this example we assume the RC file is saved in the current directory under the name "admin-openrc.sh".

2.3.4 4. Preparation steps with OpenStack

In the VMTP virtual environment, you can run any OpenStack CLI command (since the virtual environment will have all standard OpenStack client packages installed along with VMTP). Source the RC file so we can execute the CLI commands:

```
. admin-openrc.sh
```

4.1. Verify flavor names

If you are planning to reuse an existing flavor, we will have to check the flavor names available to select one flavor that VMTP should use to launch VM instances. List the flavors (results may be different):

```
$ nova flavor-list
+----+-----------+-----------+------+-----------+------+-------+-------------+--------
↪---+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor | Is_
↪Public |
+----+-----------+-----------+------+-----------+------+-------+-------------+--------
↪---+
| 1  | m1.tiny   | 512       | 1    | 0         |      | 1     | 1.0         | True  ␣
↪   |
| 2  | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0         | True  ␣
↪   |
| 3  | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0         | True  ␣
↪   |
| 4  | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0         | True  ␣
↪   |
```

```
| 5  | m1.xlarge | 16384     | 160  | 0          |      | 8     | 1.0          | True  ␣
↪   |
+----+-----------+-----------+------+------------+------+-------+--------------+--------
↪---+
$
```

Otherwise, a temporary flavor can be created for you from the config file specification (Step 5).

4.2. Upload any Linux VM image to OpenStack

VMTP requires a standard Linux VM image to run its tests in OpenStack. You can skip this step if you already have a standard Linux VM image in your OpenStack (Ubuntu, Fedora, RHEL. . .).

Otherwise, you can upload any Linux VM image using the glance CLI or using the Horizon dashboard. In the example below we will upload the Ubuntu 14.04 cloud image available from the uec-images.ubuntu.com web site using the glance CLI and we will name it "Ubuntu Server 14.04".

If your OpenStack can access directly the Internet:

```
glance --os-image-api-version 1 image-create --copy-from http://uec-images.ubuntu.com/
↪trusty/current/trusty-server-cloudimg-amd64-uefi1.img --disk-format qcow2 --
↪container-format bare --name 'Ubuntu Server 14.04'
```

The glance command will return immediately but it will take some time for the file to get transferred. You will need to check for the status of the image before you can use it (will "queued", then "saving" then "active" if there is no issue).

If you prefer to make a local copy of the image:

```
wget http://uec-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-uefi1.
↪img
```

Then copy it to OpenStack using the glance CLI:

```
glance --os-image-api-version 1 image-create --file /tmp/vmtp/trusty-server-cloudimg-
↪amd64-uefi1.img --disk-format qcow2 --container-format bare --name 'Ubuntu Server␣
↪14.04'
```

Then list the images to verify:

```
$ glance image-list
+--------------------------------------+---------------------+
| ID                                   | Name                |
+--------------------------------------+---------------------+
| 5d7899d9-811c-483f-82b3-282a9bf143bf | cirros              |
| 443ee290-b714-4bfe-9acb-b996ed6cc118 | Ubuntu 14.04        |
+--------------------------------------+---------------------+
$ glance image-show 443ee290-b714-4bfe-9acb-b996ed6cc118
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | 479a314d90cefc163fdcfb875a070cd8     |
| container_format | bare                                 |
| created_at       | 2016-07-04T17:53:20Z                 |
| disk_format      | qcow2                                |
| id               | 443ee290-b714-4bfe-9acb-b996ed6cc118 |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
```

```
| name              | Ubuntu 14.04                         |
| owner             | 5d912149f7474804824a463464874a21     |
| protected         | False                                |
| size              | 268829184                            |
| status            | active                               |
| tags              | []                                   |
| updated_at        | 2016-07-04T18:06:38Z                 |
| virtual_size      | None                                 |
| visibility        | private                              |
+-------------------+--------------------------------------+
$
```

2.3.5 5. Create your VMTP config file

Get a copy of the default VMTP configuration file and save it in the local directory as "vmtp.cfg":

```
cp cfg.default.yaml vmtp.cfg
```

Edit the vmtp.cfg file and make sure the following parameters are set properly:

- "image_name" must be the image name to use by VMTP ('Ubuntu Server 14.04' in the above example)
- "ssh_vm_username" must be a valid user name for the Linux image ("ubuntu" for Ubuntu images)
- "flavor_type" must be either an appropriate flavor name (step 4.1 above) or a custom flavor will be created with "flavor_type" name and specification declared in "flavor" config.
- "flavor" must be the specification of a custom flavor that will be created in case "flavor_type" is non-existing in OpenStack.

2.3.6 6. Run VMTP

VMTP options used:

- -d : debug mode (more verbose)
- -c vmtp.cfg : specify the config file to use
- -r admin-openrc.sh : specify the RC file to use
- -p secret : specify the OpenStack password to use (replace with your own password)
- –protocol T : only do TCP throughput test (shorter time)
- –json test.json : save results in json format to a file

```
python vmtp.py -d -c vmtp.cfg -r admin-openrc.sh -p secret --protocol T --json test.
↪json
```

This should produce an output similar to this (a complete run with the above options should take around 15 minutes but may vary based on the control plane speed of your OpenStack cloud):

```
Using http://172.29.86.28:5000/v2.0
VM public key:  /home/localadmin/.ssh/id_rsa.pub
VM private key: /home/localadmin/.ssh/id_rsa
Found image Ubuntu Server 14.04 to launch VM, will continue
Using external network: ext-net
Found external router: demo-router
```

```
Created internal network: pns-internal-net
Created internal network: pns-internal-net2
Ext router associated to pns-internal-net
Ext router associated to pns-internal-net2
OpenStack agent: Open vSwitch agent
OpenStack network type: vlan
[TestServer1] Creating server VM...
[TestServer1] Starting on zone nova:compute-server-2
[TestServer1] VM status=BUILD, retrying 1 of 50...
[TestServer1] VM status=BUILD, retrying 2 of 50...
...
[TestServer1] Floating IP 10.23.220.45 created
[TestServer1] Started - associating floating IP 10.23.220.45
[TestServer1] Internal network IP: 192.168.1.3
[TestServer1] SSH IP: 10.23.220.45
[TestServer1] Setup SSH for ubuntu@10.23.220.45
[TestServer1] Installing nuttcp-7.3.2...
[TestServer1] Copying nuttcp-7.3.2 to target...
[TestServer1] Starting nuttcp-7.3.2 server...
[TestServer1]
[TestClient1] Creating client VM...
[TestClient1] Starting on zone nova:compute-server-2
[TestClient1] VM status=BUILD, retrying 1 of 50...
[TestClient1] VM status=BUILD, retrying 2 of 50...
...
[TestClient1] Floating IP 10.23.220.46 created
[TestClient1] Started - associating floating IP 10.23.220.46
[TestClient1] Internal network IP: 192.168.1.4
[TestClient1] SSH IP: 10.23.220.46
[TestClient1] Setup SSH for ubuntu@10.23.220.46
[TestClient1] Installing nuttcp-7.3.2...
[TestClient1] Copying nuttcp-7.3.2 to target...
============================================================
Flow 1: VM to VM same network fixed IP (intra-node)
[TestClient1] Measuring TCP Throughput (packet size=65536)...
[TestClient1] /tmp/nuttcp-7.3.2 -T10  -l65536 -p5001 -P5002 -fparse 192.168.1.3
[TestClient1] megabytes=20329.1875 real_seconds=10.00 rate_Mbps=17049.6212 tx_cpu=92␣
↪rx_cpu=53 retrans=0 rtt_ms=0.47
...
{   'az_from': u'nova:compute-server-2',
    'az_to': u'nova:compute-server-2',
    'desc': 'VM to VM same network fixed IP (intra-node)',
    'distro_id': 'Ubuntu',
    'distro_version': '14.04',
    'ip_from': u'192.168.1.4',
    'ip_to': u'192.168.1.3',
    'results': [   {   'pkt_size': 65536,
                       'protocol': 'TCP',
                       'rtt_ms': 0.47,
                       'throughput_kbps': 17458812,
                       'tool': 'nuttcp-7.3.2'},
                   {   'pkt_size': 65536,
                       'protocol': 'TCP',
                       'rtt_ms': 0.19,
                       'throughput_kbps': 13832383,
                       'tool': 'nuttcp-7.3.2'},
                   {   'pkt_size': 65536,
                       'protocol': 'TCP',
```

```
                      'rtt_ms': 0.21,
                      'throughput_kbps': 17130867,
                      'tool': 'nuttcp-7.3.2'}]}
[TestClient1] Floating IP 10.23.220.46 deleted
[TestClient1] Instance deleted
[TestClient2] Creating client VM...
[TestClient2] Starting on zone nova:compute-server-2
[TestClient2] VM status=BUILD, retrying 1 of 50...
[TestClient2] VM status=BUILD, retrying 2 of 50...

...

---- Cleanup ----
[TestServer1] Terminating nuttcp-7.3.2
[TestServer1] Floating IP 10.23.220.45 deleted
[TestServer1] Instance deleted
Network pns-internal-net deleted
Network pns-internal-net2 deleted
Removed public key pns_public_key
Deleting security group

Summary of results
==================
Total Scenarios:   22
Passed Scenarios:  5 [100.00%]
Failed Scenarios:  0 [0.00%]
Skipped Scenarios: 17
+----------+-----------------------------------------------------+--------------------+---
↪-----------------------------------------+
| Scenario | Scenario Name                                       | Functional Status |␣
↪Data                                      |
+----------+-----------------------------------------------------+--------------------+---
↪-----------------------------------------+
| 1.1      | Same Network, Fixed IP, Intra-node, TCP             | PASSED             | {
↪'tp_kbps': '16140687', 'rtt_ms': '0.29'}    |
| 1.2      | Same Network, Fixed IP, Intra-node, UDP             | SKIPPED            | {}
↪                                          |
| 1.3      | Same Network, Fixed IP, Intra-node, ICMP            | SKIPPED            | {}
↪                                          |
| 2.1      | Same Network, Fixed IP, Inter-node, TCP             | PASSED             | {
↪'tp_kbps': '4082749', 'rtt_ms': '0.5'}      |
| 2.2      | Same Network, Fixed IP, Inter-node, UDP             | SKIPPED            | {}
↪                                          |
| 2.3      | Same Network, Fixed IP, Inter-node, ICMP            | SKIPPED            | {}
↪                                          |
| 3.1      | Different Network, Fixed IP, Intra-node, TCP        | PASSED             | {
↪'tp_kbps': '2371753', 'rtt_ms': '0.386667'} |
| 3.2      | Different Network, Fixed IP, Intra-node, UDP        | SKIPPED            | {}
↪                                          |
| 3.3      | Different Network, Fixed IP, Intra-node, ICMP       | SKIPPED            | {}
↪                                          |
| 4.1      | Different Network, Fixed IP, Inter-node, TCP        | PASSED             | {
↪'tp_kbps': '2036303', 'rtt_ms': '0.623333'} |
| 4.2      | Different Network, Fixed IP, Inter-node, UDP        | SKIPPED            | {}
↪                                          |
| 4.3      | Different Network, Fixed IP, Inter-node, ICMP       | SKIPPED            | {}
↪                                          |
| 5.1      | Different Network, Floating IP, Intra-node, TCP     | PASSED             | {
↪'tp_kbps': '2260145', 'rtt_ms': '0.476667'} |
```

```
| 5.2      | Different Network, Floating IP, Intra-node, UDP  | SKIPPED           | {}
↪                                                  |
| 5.3      | Different Network, Floating IP, Intra-node, ICMP | SKIPPED           | {}
↪                                                  |
| 6.1      | Different Network, Floating IP, Inter-node, TCP  | PASSED            | {
↪'tp_kbps': '2134303', 'rtt_ms': '0.543333'} |
| 6.2      | Different Network, Floating IP, Inter-node, UDP  | SKIPPED           | {}
↪                                                  |
| 6.3      | Different Network, Floating IP, Inter-node, ICMP | SKIPPED           | {}
↪                                                  |
| 7.1      | Native Throughput, TCP                           | SKIPPED           | {}
↪                                                  |
| 7.2      | Native Throughput, UDP                           | SKIPPED           | {}
↪                                                  |
| 7.3      | Native Throughput, ICMP                          | SKIPPED           | {}
↪                                                  |
| 8.1      | VM to Host Uploading                             | SKIPPED           | {}
↪                                                  |
| 8.2      | VM to Host Downloading                           | SKIPPED           | {}
↪                                                  |
+----------+--------------------------------------------------+-------------------+---
↪-----------------------------------------+
Saving results in json file: test.json...
```

2.3.7 8. Generate the results chart from the JSON result file

Assuming the json result file is saved by the container run the vmtp_genchart container command from the host current directory:

```
$ python vmtp_genchart.py -c test.html test.json
Generating chart drawing code to /tmp/vmtp/test.html...
$
```

vmtp_genchart options:

- -c test.html : save the generated html file to the mapped directory
- test.json : the json file that contains the results of the VMTP run

The fie is available in the current directory and can be viewed with any browser:

```
$ ls -l test.html
-rw-r--r-- 1 root root 1557 Jul  4 14:10 test.html
```

Installation and Quick Start Guides:

- *VMTP Docker Container Installation and Quick Start Guide*
- *VMTP PyPI Installation and Quick Start Guide*
- *VMTP Git Installation and Quick Start Guide*

Note: Installation from Docker container or PyPI will only have the latest stable version.

CHAPTER 3

Usage

3.1 VMTP Usage

```
usage: vmtp [-h] [-c <config_file>] [-sc] [-r <openrc_file>]
            [-m <gmond_ip>[:<port>]] [-p <password>] [-t <time>]
            [--host <user>@<host_ssh_ip>[:<password>:<server-listen-if-name>]]
            [--external-host <user>@<host_ssh_ip>[:password>]]
            [--controller-node <user>@<host_ssh_ip>[:<password>]]
            [--mongod-server <server ip>] [--json <file>]
            [--tp-tool <nuttcp|iperf>]
            [--availability_zone <availability_zone>] [--hypervisor [<az>:]
            <hostname>] [--inter-node-only] [--same-network-only]
            [--protocols <T|U|I>] [--bandwidth <bandwidth>]
            [--tcpbuf <tcp_pkt_size1,...>] [--udpbuf <udp_pkt_size1,...>]
            [--reuse_network_name <network_name>]
            [--os-dataplane-network <network_name>] [--delete-image-after-run]
            [--no-env] [--vnic-type <direct|macvtap|normal>] [-d] [-v]
            [--stop-on-error] [--vm-image-url <url_to_image>]
            [--test-description <test_description>]

optional arguments:
  -h, --help            show this help message and exit
  -c <config_file>, --config <config_file>
                        override default values with a config file
  -sc, --show-config    print the default config
  -r <openrc_file>, --rc <openrc_file>
                        source OpenStack credentials from rc file
  -m <gmond_ip>[:<port>], --monitor <gmond_ip>[:<port>]
                        Enable CPU monitoring (requires Ganglia)
  -p <password>, --password <password>
                        OpenStack password
  -t <time>, --time <time>
                        throughput test duration in seconds (default 10 sec)
  --host <user>@<host_ssh_ip>[:<password>:<server-listen-if-name>]
```

```
                      native host throughput (password or public key
                      required)
--external-host <user>@<host_ssh_ip>[:password>]
                      external-VM throughput (password or public key
                      required)
--controller-node <user>@<host_ssh_ip>[:<password>]
                      controller node ssh (password or public key required)
--mongod-server <server ip>
                      provide mongoDB server IP to store results
--json <file>         store results in json format file
--tp-tool <nuttcp|iperf>
                      transport perf tool to use (default=nuttcp)
--availability_zone <availability_zone>
                      availability zone for running VMTP
--hypervisor [<az>:] <hostname>
                      hypervisor to use (1 per arg, up to 2 args)
--inter-node-only     only measure inter-node
--same-network-only   only measure same network
--protocols <T|U|I>   protocols T(TCP), U(UDP), I(ICMP) - default=TUI (all)
--bandwidth <bandwidth>
                      the bandwidth limit for TCP/UDP flows in K/M/Gbps,
                      e.g. 128K/32M/5G. (default=no limit)
--tcpbuf <tcp_pkt_size1,...>
                      list of buffer length when transmitting over TCP in
                      Bytes, e.g. --tcpbuf 8192,65536. (default=65536)
--udpbuf <udp_pkt_size1,...>
                      list of buffer length when transmitting over UDP in
                      Bytes, e.g. --udpbuf 128,2048. (default=128,1024,8192)
--reuse_network_name <network_name>
                      the network to be reused for performing tests
--os-dataplane-network <network_name>
                      Internal network name for OpenStack to hold data plane
                      traffic
--delete-image-after-run
                      delete image that are uploaded by VMTP when tests are
                      finished
--no-env              do not read env variables
--vnic-type <direct|macvtap|normal>
                      binding vnic type for test VMs
-d, --debug           debug flag (very verbose)
-v, --version         print version of this script and exit
--stop-on-error       Stop and keep everything as-is on error (must cleanup
                      manually)
--vm-image-url <url_to_image>
                      URL to a Linux image in qcow2 format that can be
                      downloaded fromor location of the image file with
                      prefix file://
--test-description <test_description>
                      The test description to be stored in JSON or MongoDB
```

3.1.1 Configuration File

VMTP configuration files follow the yaml syntax and contain variables used by VMTP to run and collect performance data. The default configuration is stored in the cfg.default.yaml file.

Default values should be overwritten for any cloud under test by defining new variable values in a new configuration file that follows the same format. Variables that are not defined in the new configuration file will retain their default

values.

The precedence order for configuration files is as follows: - the command line argument "-c <file>" has highest precedence - $HOME/.vmtp.yaml if the file exists in the user home directory - cfg.default.yaml has the lowest precedence (always exists in the VMTP package root directory)

To override a default value set in cfg.default.yaml, simply redefine that value in the configuration file passed in -c or in the $HOME/.vmtp.yaml file. Check the content of cfg.default.yaml file as it contains the list of configuration variables and instructions on how to set them.

Note: The configuration file is not needed if VMTP only runs the native host throughput option (*–host*)

3.1.2 OpenStack openrc File

VMTP requires downloading an "openrc" file from the OpenStack Dashboard (Project|Acces&Security!Api Access|Download OpenStack RC File)

This file should then be passed to VMTP using the *-r* option or should be sourced prior to invoking VMTP.

Note: The openrc file is not needed if VMTP only runs the native host throughput option (*–host*)

3.1.3 Access Info for Controller Node

By default, VMTP is not able to get the Linux distro nor the OpenStack version of the cloud deployment under test. However, by providing the credentials of the controller node under test, VMTP will try to fetch these information, and output them along in the JSON file or to the MongoDB server. For example to retrieve the OpenStack distribution information on a given controller node:

```
python vmtp.py --json tb172.json --test-description 'Testbed 172' --controller-node␣
↪root@172.22.191.172
```

3.1.4 Bandwidth Limit for TCP/UDP Flow Measurements

Specify a value in *–bandwidth* will limit the bandwidth when performing throughput tests.

The default behavior for both TCP/UDP are unlimited. For TCP, we are leveraging on the protocol itself to get the best performance; while for UDP, we are doing a binary search to find the optimal bandwidth.

This is useful when running vmtp on production clouds. The test tool will use up all the bandwidth that may be needed by any other live VMs if we don't set any bandwidth limit. This feature will help to prevent impacting other VMs while running the test tool.

3.1.5 Host Selection and Availability Zone

VMTP requires 1 physical host to perform intra-node tests and 2 hosts to perform inter-node tests. There are multiple ways to specify the placement of test VMs to VMTP. By default, VMTP will pick the first 2 compute hosts it can find, regardless of the availability zone.

It is possible to limit the host selection to a specific availability zone by specifying its name in the yaml configuration file ('availability_name' parameter).

The *–hypervisor* argument can also be used to specify explicitly on which hosts to run the test VMs. The first *–hypervisor* argument specifies on which host to run the test server VM. The second *–hypervisor* argument (in the command line) specifies on which host to run the test client VMs.

The syntax to use for the argument value is either availability_zone and host name separated by a column (e.g. "–hypervisor nova:host26") or host name (e.g. "–hypervisor host12"). In the latter case, VMTP will automaticaly pick the availability zone of the host.

Picking a particular host can be handy for example when exact VM placement can impact the data path performance (for example rack based placement).

The value of the argument must match the hypervisor host name as known by OpenStack (or as displayed using "nova hypervisor-list").

If an availability zone is provided, VMTP will check that the host name exists in that availability zone.

3.1.6 Upload Images to Glance

VMTP requires a Linux image available in Glance to spawn VMs. It could be uploaded manually through Horizon or CLI, or VMTP will try to upload the image defined in the configuration file automatically.

There is a candidate image defined in the default config already. It has been verified working, but of course it is OK to try other Linux distro as well.

3.1.7 VNIC Type and SR-IOV Support

By default test VMs will be created with ports that have a "normal" VNIC type. To create test VMs with ports that use PCI passthrough SR-IOV, specify **–vnic_type direct**. This will assume that the host where the VM are instantiated have SR-IOV capable NIC.

An exception will be thrown if a test VM is lauched on a host that does not have SRIOV capable NIC or has not been configured to use such feature.

3.1.8 Provider Network

Provider networks are created by network administrators, which specifies the details of how network is physically realized, and usually match some existing networks in the data center. In general, provider networks only handle layer-2 connectivity for instances, and lack the concept of fixed and floating IP addresses. Running VMTP on provider network, means that VMTP is directly running on the infrastructure VLAN and no L3 in OpenStack is involved (L3 will be handled in the legacy L3 Router).

VMTP supports to run on a provider network by supplying the provider network name via either CLI or config file with **–reuse_network_name <network_name>**. Just happens that this mode allows to run on real provider network, but also the regular neutron network settings (e.g. VxLAN) where the user would like VMTP to reuse an already created network. VMTP will perform the L2-only throughput tests on the network provided when this parameter is set.

Note that the fixed IP addresses assigned to the test VM on the provider network must be reachable from the host which VMTP application is running on.

3.2 Examples of running VMTP on an OpenStack Cloud

Examples below invoke VMTP using the python interpreter (suitable for git installation). For Docker container or pip installation, simply invoke vmtp directly (instead of "python vmtp.py").

3.2.1 Example 1: Typical Run

Run VMTP on an OpenStack cloud with the default configuration file, use "admin-openrc.sh" as the rc file, and "admin" as the password:

```
python vmtp.py -r admin-openrc.sh -p admin
```

This will generate 6 standard sets of performance data: (1) VM to VM same network (intra-node, private fixed IP) (2) VM to VM different network (intra-node, L3 fixed IP) (3) VM to VM different network and tenant (intra-node, floating IP) (4) VM to VM same network (inter-node, private fixed IP) (5) VM to VM different network (inter-node, L3 fixed IP) (6) VM to VM different network and tenant (inter-node, floating IP)

By default, the performance data of all three protocols (TCP/UDP/ICMP) will be measured for each scenario mentioned above. However, it can be overridden by providing *–protocols*:

```
python vmtp.py -r admin-openrc.sh -p admin --protocols IT
```

This will tell VMTP to only collect ICMP and TCP measurements.

3.2.2 Example 2: Cloud upload/download performance measurement

Run VMTP on an OpenStack cloud with a specified configuration file (mycfg.yaml), and saved the result to a JSON file:

```
python vmtp.py -c mycfg.yaml -r admin-openrc.sh -p admin --external-host␣
↪localadmin@172.29.87.29 --json res.json
```

This run will generate 8 sets of performance data, the standard 6 sets mentioned above, plus two sets of upload/download performance data for both TCP and UDP. If you do not have ssh password-less access to the external host (public key) you must specify a password:

```
python vmtp.py -c mycfg.yaml -r admin-openrc.sh -p admin --external-host␣
↪localadmin@172.29.87.29:secret --json res.json
```

3.2.3 Example 3: Retrieve the OpenStack deployment details

Run VMTP on an OpenStack cloud, fetch the details of the deployment and store it to JSON file. Assume the controller node is on 192.168.12.34 with admin/admin:

```
python vmtp.py -r admin-openrc.sh -p admin --json res.json --controller-node root@192.
↪168.12.34:admin
```

In addition, VMTP also supports to store the results to a MongoDB server:

```
python vmtp.py -r admin-openrc.sh -p admin --json res.json --mongod-server 172.29.87.
↪29 --controller-node root@192.168.12.34:admin
```

Before storing info into MongoDB, some configurations are needed to change to fit in your environment. By default, VMTP will store to database "client_db" with collection name "pns_web_entry", and of course these can be changed in the configuration file. Below are the fields which are related to accessing MongoDB:

```
vmtp_mongod_port
vmtp_db
vmtp_collection
```

3.2.4 Example 4: Specify which compute nodes to spawn VMs

Run VMTP on an OpenStack cloud, spawn the test server VM on tme212, and the test client VM on tme210. Save the result, and perform the inter-node measurement only:

```
python vmtp.py -r admin-openrc.sh -p admin --inter-node-only --json res.json --
↪hypervisor tme212 --hypervisor tme210
```

3.2.5 Example 5: Collect native host performance data

Run VMTP to get native host throughput between 172.29.87.29 and 172.29.87.30 using the localadmin ssh username and run each tcp/udp test session for 120 seconds (instead of the default 10 seconds):

```
python vmtp.py --host localadmin@172.29.87.29 --host localadmin@172.29.87.30 --time␣
↪120
```

The first IP passed (*–host*) is always the one running the server side. If you do not have public keys setup on these targets, you must provide a password:

```
python vmtp.py --host localadmin@172.29.87.29:secret --host localadmin@172.29.87.
↪30:secret --time 120
```

It is also possible to run VMTP between pre-existing VMs that are accessible through SSH (using floating IP) if you have the corresponding private key to access them.

In the case of servers that have multiple NIC and IP addresses, it is possible to specify the server side listening interface name to use (if you want the client side to connect using the associated IP address) For example, to measure throughput between 2 hosts using the network attached to the server interface "eth5":

```
python vmtp.py --host localadmin@172.29.87.29::eth5 --host localadmin@172.29.87.30
```

3.2.6 Example 6: IPV6 throughput measurement

It is possible to use VMTP to measure throughput for IPv6.

Set ipv6_mode to slaac, dhcpv6-stateful or dhcpv6-stateless. If SLAAC or DHCPv6 stateless is enabled make sure to have radvd packaged in as part of openstack install. For DHCPv6 stateful you need dnsmasq version >= 2.68. The test creates 2 networks and creates 1 IPv4 and 1 IPv6 subnet inside each of these networks. The subnets are created based on the IPv6 mode that you set in the configuration file. The Floating IP result case is skipped for IPv6 since there is no concept of a floating ip with IPv6.

3.3 Running VMTP as a library

VMTP supports to be invoked from another Python program, just like an API call. Once the benchmarking is finished, the API will return a Python dictionary with all details.

Example of code for running VMTP as an API call:

```
import argparse
opts = argparse.Namespace()
opts.rc = "<path_to_rc_file>"
opts.passwd = "<password_of_the_cloud>"
opts.inter_node_only = True
```

```
opts.json = "my.json"

import vmtp
vmtp.run_vmtp(opts)
```

3.4 Generating charts from JSON results

```
usage: vmtp_genchart.py [-h] [-c <file>] [-b] [-p <all|tcp|udp>] [-v]
                        <file> [<file> ...]

VMTP Chart Generator V0.0.1

positional arguments:
  <file>                vmtp json result file

optional arguments:
  -h, --help            show this help message and exit
  -c <file>, --chart <file>
                        create and save chart in html file
  -b, --browser         display (-c) chart in the browser
  -p <all|tcp|udp>, --protocol <all|tcp|udp>
                        select protocols:all, tcp, udp
  -v, --version         print version of this script and exit
```

Examples of use:

Generate charts from the JSON results file "tb172.json", store resulting html to "tb172.html" and open that file in the browser:

```
python vmtp_genchart.py --chart tb172.html --browser tb172.json
```

Same but only show UDP numbers:

```
python vmtp_genchart.py --chart tb172.html --browser --protocol udp tb172.json
```

CHAPTER 4

Setup

4.1 SSH Authentication

VMTP can optionally SSH to the following hosts: - OpenStack controller node (if the –controller-node option is used) - External host for cloud upload/download performance test (if the –external-host option is used) - Native host throughput (if the –host option is used)

To connect to these hosts, the SSH library used by VMTP will try a number of authentication methods: - if provided at the command line, try the provided password (e.g. –controller-node localadmin@10.1.1.78:secret) - user's personal private key (~/.ssh/id_rsa) - if provided in the configuration file, a specific private key file (private_key_file variable)

SSH to the test VMs is always based on key pairs with the following precedence: - if provided in the passed configuration file, use the configured key pair (private_key_file and public_key_file variables), - otherwise use the user's personal key pair (~/.ssh/id_rsa and ~/.ssh/id_rsa.pub) - otherwise if there is no personal key pair configured, create a temporary key pair to access all test VMs

To summarize: - if you have a personal key pair configured in your home directory, VMTP will use that key pair for all SSH connections (including to the test VMs) - if you want to use your personal key pair, there is nothing to do other than making sure that the targeted hosts have been configured with the associated public key

In any case make sure you specify the correct username. If there is a problem, you should see an error message and stack trace after the SSH library times out.

CHAPTER 5

Implementation

5.1 TCP Throughput Measurement

The TCP throughput reported is measured using the default message size of the test tool (64KB with nuttcp). The TCP MSS (maximum segment size) used is the one suggested by the TCP-IP stack (which is dependent on the MTU).

5.2 UDP Throughput Measurement

UDP throughput is tricky because of limitations of the performance tools used, limitations of the Linux kernel used and criteria for finding the throughput to report.

The default setting is to find the "optimal" throughput with packet loss rate within the 2%~5% range. This is achieved by successive iterations at different throughput values.

In some cases, it is not possible to converge with a loss rate within that range and trying to do so may require too many iterations. The algorithm used is empiric and tries to achieve a result within a reasonable and bounded number of iterations. In most cases the optimal throughput is found in less than 30 seconds for any given flow.

Note: UDP measurements are only available with nuttcp (not available with iperf).

CHAPTER 6

Caveats and Known Issues

- UDP throughput is not available if iperf is selected (the iperf UDP reported results are not reliable enough for iterating)
- If VMTP hangs for native hosts throughputs, check firewall rules on the hosts to allow TCP/UDP ports 5001 and TCP port 5002
- When storing the results to JSON or MongoDB, the quotes in the command-line will not be saved. In a unix-like environment, the magic happened even before Python can see them. e.g. quotes get consumed, variables get interpolated, etc. Keep this in mind when you want to execute the command stored in "*args*", and pay more attention in any parameter that may have quotes inside like *test_description*.

CHAPTER 7

Contributing

7.1 Contribute to VMTP

Below are a simplified version of the workflow to work on VMTP. For complete instructions, you have to follow the Developer's Guide in OpenStack official documents. Refer to *below section* for links.

7.1.1 Start working

Before starting, a GitHub/StackForge respository based installation must be done. Refer *here* for detailed documentation.

1. From the root of your workspace, check out a new branch to work on:

```
$ git checkout -b <TOPIC-BRANCH>
```

2. Happy working on your code for features or bugfixes;

7.1.2 Before Commit

There are some criteria that are enforced to commit to VMTP. Below commands will perform the check and make sure your code complys with it.

3. PEP 8:

```
$ tox -epep8
```

Note: The first run usually takes longer, as tox will create a new virtual environment and download all dependencies. Once that is the done, further run will be very fast.

4. Run the test suite:

```
$ tox -epython27
```

5. If you made a documentation change (i.e. changes to .rst files), make sure the documentation is built as you expected:

```
$ cd <vmtp-ws-root>/doc
$ make html
```

Once finished, the documentation in HTML format will be ready at <vmtp-ws-root>/doc/build/html.

7.1.3 Submit Review

6. Commit the code:

```
$ git commit -a
```

Note: For a feature commit, please supply a clear commit message indicating what the feature is; for a bugfix commit, please also containing a launchpad link to the bug you are working on.

7. Submit the review:

```
$ git review <TOPIC-BRANCH>
```

The members in the VMTP team will get notified once the Jenkin verification is passed. So watch your email from the review site, as it will contain the updates for your submission.

8. If the code is approved with a +2 review, Gerrit will automatically merge your code.

7.2 File Bugs

Bugs should be filed on Launchpad, not GitHub:

https://bugs.launchpad.net/vmtp

7.3 Build the VMTP Docker Image

The VMTP Docker images are published in the DockerHub berrypatch repository: https://hub.docker.com/r/berrypatch/vmtp/

Two files are used to build the Docker image: *Dockerfile* and *.dockerignore*. The former provides all the build instructions while the latter provides the list of files/directories that should not be copied to the Docker image.

In order to make the Docker image clean, remove all auto generated files from the root of your workspace first. It is strongly recommeneded to simply pull a new one from GitHub/OPenStack. Specify the image name and the tag, and feed them to docker build. The tag should be an existing git tag for the vmtp repository.

To build the image with tag "2.0.0" and "latest":

```
$ cd <vmtp-ws-root>
$ sudo docker build --tag=berrypatch/vmtp:2.0.0 .
$ sudo docker build --tag=berrypatch/vmtp:latest .
```

The images should be available for use:

```
$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             ␣
↪VIRTUAL SIZE
berrypatch/vmtp     2.0.0               9f08056496d7        27 hours ago        494.6␣
↪MB
berrypatch/vmtp     latest              9f08056496d7        27 hours ago        494.6␣
↪MB
```

For exchanging purposes, the image could be saved to a tar archive. You can distribute the VMTP Docker image among your servers easily with this feature:

```
$ sudo docker save -o <IMAGE_FILE> <IMAGE_ID>
```

To publish you need to be a member of the berrypatch vmtp team. After the login (requires your DockerHub username and password), push the appropriate version to berrypatch:

```
$ sudo docker login
$ sudo docker push berrypatch/vmtp:2.0.0
$ sudo docker push berrypatch/vmtp:latest
```

7.4 Developer's Guide of OpenStack

If you would like to contribute to the development of OpenStack, you must follow the steps in this page:

https://docs.openstack.org/infra/manual/developers.html

Once those steps have been completed, changes to OpenStack should be submitted for review via the Gerrit tool, following the workflow documented at:

https://docs.openstack.org/infra/manual/developers.html#development-workflow

Pull requests submitted through GitHub will be ignored.