Yardstick Project Update: Python 3 support and VNF Characterization

Ross Brattain, Deepak S.
Intel
Agenda

• Lessons learned from adding Python 3 support in Yardstick
  • Follow the OpenStack Python 3 guidelines
  • Use automated tools for code conversion
  • Convert in stages

• How to write a VNF characterization test in Yardstick
  • Network Services Benchmarking extensions
  • VNF descriptors and topology
  • Traffic generators and KPI collection
Follow the OpenStack guidelines

• https://wiki.openstack.org/wiki/Python3

• Pay attention to JSON and Unicode byte/string decode/encode
  • string → encode() → I/O → bytes → decode() → string

• Use the Oslo JSON encode/decode wrappers
  • OpenStack added them to Oslo to ease compatibility with Python2/3
  • Use the OpenStack libraries as much as possible
  • Replace json.dumps(obj) with oslo_serialization.jsonutils.dump_as_bytes(obj)
  • Replace json.loads(obj) with oslo_serialization.jsonutils.loads(obj)
Use automated tools for code conversion

• Use automated tools for code conversion
  • Python-modernize
  • sixer
  • Pylint
  • Pycharm code compatibility inspections
• Python-modernize can be extended to do other automated Python code changes
Python 3 conversion stages

https://etherpad.opnfv.org/p/AddingPython3Support

• Use automatic conversion to bring the code up to Python 2.7 latest
• Fix Python 2 unittests
• Convert to Python 3
• Fix Python 3 unittests
• Run with Python 3
• Fix Unicode issues
VNF characterization and Network Services Benchmarking

- Network Services Benchmarking extensions
- Test Topology and VNF descriptors
- Traffic generators and KPI collection
4.7 NS Under Test

For pre-deployment validation of a Network Service (NS), the NS represents the SUT.

The target requirement is to test Network Services consisting of multiple VNFs
Review of existing Yardstick framework architecture

There is a lot of implicit network complexity in this diagram.

For initial requirements we are focusing on PNF to VNF with 2 ports.
Architecture of NSB extensions to Yardstick

We require minimum 3 ports per node.
Note that in some cases mgmt control-plane and dataplane share bridges.

OpenStack Network Topology

This is one possible network topology. It illustrates the complexity required by OpenStack. The topology cannot be defined by Yardstick. Yardstick can only use the existing topology created by the installer.

In this case in order for the Traffic Generator to access both VNFs it needs to be trunked and insert its own VLAN tags.

In this case we use provider networks to bypass the Neutron router bottleneck.
NetworkServiceTestCase sequence diagram part 1

https://wiki.opnfv.org/display/yardstick/NSB+sequence+diagram
NetworkServiceTest sequence diagram part 2

https://wiki.opnfv.org/display/yardstick/NSB+sequence+diagram
Testcase → Topology → VNF descriptor

The YAML files are linked by relative pathnames

Testcase

```
schema: yardstick:task:0.1
scenarios:
  - type: NSPerf
topology: vpe_vnf_topology.yaml
```

Topology

```
nsd:nsd-catalog:
  nsd:
    - id: VPE
      name: VPE
      short-name: VPE
      description: scenario with VPE,L3fwd and VNF
    constituent-vnfd:
      - member-vnf-index: '1'
        vnfd-id-ref: tg__1
        VNF model: ../../vnf_descriptors/tg_rfc2544_tpl.yaml
      - member-vnf-index: '2'
        vnfd-id-ref: vnf__1
```

VNF descriptor

```
vnfd:vnfd-catalog:
  vnfd:
    - id: VpeApproxVnf
      name: VpeVnfSshIntel
      short-name: VpeVnf
      description: vPe approximation using DPDK
      vdu:
        - id: vpevnf-baremetal
          name: vpevnf-baremetal
          description: vpe approximation using DPDK
          external-interface:
            - name: xe0
              virtual-interface:
                type: PCI-PASSTHROUGH
                vnfd-connection-point-ref: xe0
                routing_table: {{ routing_table }}
```

VNF model:

```
VNF model: ../../vnf_descriptors/vpe_vnf.yaml
```
VNF Descriptor

• YAML template designed based on the vnfd:vdu, vld type standards.
• Eventually will take TOSCA or other VNFD input
• Each VNF needs a descriptor
• Can have static information, or information can be overridden dynamically based on Heat template outputs or Neutron network info
Sample VNF Descriptor

```python
vnfd:vnfd-catalog:
  vnfd:
    - id: VpeApproxVnf
      name: VpeVnfSShIntel
      short-name: VpeVnf
      description: vPe approximation using DPDK
      mgmt-interface:
        vdu-id: vpevnf-baremetal
        user: '{{user}}'
        password: '{{password}}'
        ip: '{{ip}}'
        key_filename: '{{key_filename}}'
      connection-point:
        - name: xe0
          type: VPORT
        - name: xe1
          type: VPORT

vdu:
  - id: vpevnf-baremetal
    name: vpevnf-baremetal
    description: vpe approximation using DPDK
    external-interface:
      - name: xe0
        virtual-interface:
          type: PCI-PASSTHROUGH
          vpci: '{{interfaces.xe0.vpci}}'
          local_ip: '{{interfaces.xe0.local_ip}}'
          dst_ip: '{{interfaces.xe0.dst_ip}}'
          local_mac: '{{interfaces.xe0.local_mac}}'
          netmask: '{{interfaces.xe0.netmask}}'
          dst_mac: '{{interfaces.xe0.dst_mac}}'
          bandwidth: 10 Gbps

vnfd-connection-point-ref: xe0

routing_table: {{ routing_table }}

benchmark:
  kpi:
    - packets_in
    - packets_fwd
    - packets_dropped
```

The highlighted area, external-interface list is the important area.
This defines the network interface information presented to the VNF application and the Traffic Generator.

We do not have a defined schema, but minimum requirements are local/remote MAC addresses and IP addresses.

VPCI is needed for DPDK binding/unbinding, but VPCI can be auto-detected on Linux nodes.

Bandwidth and type are not used currently.
Topo：“nsd-catalog”：
  “nsd”：
    id: VPE
    name: VPE
    short-name: VPE
    description: scenario with VPE, L3fwd and VNF

“constituent-vnfd”：
  - member-vnf-index: '1'
    vnfd-id-ref: tg_1
    VNF model: ../../vnf_descriptors/tg_rfc2544_tpl.yaml
  - member-vnf-index: '2'
    vnfd-id-ref: vnf_1
    VNF model: ../../vnf_descriptors/vpe_vnf.yaml

VNF model links to VNF descriptor YAML template

vld:id maps to Heat Neutron network using vld_id meta-data

Port pairs are defined as “vnfd-connection-point-ref”:

- id: private
  name: tg_1 to vnf_1 link 1
  type: ELAN
  vnfd-connection-point-ref:
    - member-vnf-index-ref: '1'
      vnfd-connection-point-ref: xe0
      vnfd-id-ref: tg_1
    - member-vnf-index-ref: '2'
      vnfd-connection-point-ref: xe0
      vnfd-id-ref: vnf_1

- id: public
  name: vnf_1 to tg_1 link 2
  type: ELAN
  vnfd-connection-point-ref:
    - member-vnf-index-ref: '2'
      vnfd-connection-point-ref: xe1
      vnfd-id-ref: vnf_1
    - member-vnf-index-ref: '1'
      vnfd-connection-point-ref: xe1
      vnfd-id-ref: tg_1

xe0 maps to Heat Neutron network name
Sample test case

schema: "yardstick:task:0.1"

scenarios:

- type: NSPerf
traffic_profile: "../../traffic_profiles/ipv4_throughput_vpe.yaml"
topology: vpe_vnf_topology.yaml
traffic_options:
  flow: "../../traffic_profiles/ipv4_1flow_Packets_vpe.yaml"
  imix: "../../traffic_profiles/imix_voice.yaml"

nodes:
tg__1: trafficgen_1.baremetal
vnf__1: vnf.yardstick

runner:
type: Duration
duration: 400
interval: 35

contexts:
- name: baremetal
type: Node
  file: trex-baremetal.yml
- name: yardstick
  image: yardstick-vpe
  flavor: yardstick-flavor
  user: ubuntu
placement_groups:
pgrp1:
  policy: "availability"
servers:
  vnf:
    floating_ip: true
    placement: "pgrp1"

networks:
mgmt:
cidr: '10.0.1.0/24'
external_network: "yardstick-public"
xe0:
cidr: '10.0.2.0/24'
vld_id: public
xe1:
cidr: '10.0.3.0/24'
vld_id: private

xe0 network name and vld_id match values in topology
Accessing the VNF descriptor from a Python class

```python
mgmt_interface = self.vnfd["mgmt-interface"]
self.connection = ssh.SSH.from_node(mgmt_interface)

vpci = []
port = {}

for interface in ext_intf:
    virt_intf = interface["virtual-interface"]
    vpci.append(virt_intf["vpci"])

port["src_mac"] = self._split_mac_address_into_list(virt_intf["local_mac"])
port["dest_mac"] = self._split_mac_address_into_list(virt_intf["dst_mac"])

if virt_intf["dst_mac"]:
    trex_cfg["port_info"]["name: xe0 virtual-interface:
        type: PCI-PASSTHROUGH
        vpci: '{{ interfaces.xe0.vpci }}'
        local_ip: '{{ interfaces.xe0.local_ip }}'
        dst_ip: '{{ interfaces.xe0.dst_ip }}'
        local_mac: '{{ interfaces.xe0.local_mac }}'
        netmask: '{{ interfaces.xe0.netmask }}'
        dst_mac: '{{ interfaces.xe0.dst_mac }}'
        bandwidth: 10 Gbps
```
NetworkServiceTestCase Setup

```python
def setup(self):
    # 1. Verify if infrastructure mapping can meet topology
    self.map_topology_to_infrastructure(self.context_cfg, self.topology)
    # 1a. Load VNF models
    self.vnfs = self.load_vnf_models(self.scenario_cfg, self.context_cfg)
    # 1b. Fill traffic profile with information from topology
    self.traffic_profile = self._fill_traffic_profile(self.scenario_cfg, self.context_cfg)
    # 2. Provision VNFs
    for vnf in self.vnfs:
        LOG.info("Instantiating %s", vnf.name)
        vnf.instantiate(self.scenario_cfg, self.context_cfg)

    # 3. Run experiment
    # Start Listeners first to avoid losing packets
    traffic_runners = [vnf for vnf in self.vnfs if vnf.runs_traffic]
    for traffic_gen in traffic_runners:
        traffic_gen.listen_traffic(self.traffic_profile)
    # register collector with yardstick for KPI collection.
    self.collector = Collector(self.vnfs, self.traffic_profile)
    self.collector.start()
    # Start the actual traffic
    for traffic_gen in traffic_runners:
        LOG.info("Starting traffic on %s", traffic_gen.name)
        traffic_gen.run_traffic(self.traffic_profile)
```
GenericVNF Subclasses

- GenericTrafficGen
  - collect_kpi
  - instantiate
  - listen_traffic
  - run_traffic
  - scale
  - terminate
  - verify_traffic

- ProxBase
  - collect_kpi
  - instantiate
  - scale
  - terminate
  - upload_prox_config
  - get_prox_lua
  - write_prox_config

- CgnaptApproxVnf
  - collect_kpi
  - deploy_cgnapt_vnf
  - execute_command
  - get_nfvi_type
  - get_stats_vcnapt
  - instantiate
  - scale
  - setup_vnf_environment
  - terminate
  - upload_prox_config

- AclApproxVnf
  - collect_kpi
  - deploy_acl_vnf
  - find_pci
  - find_used_drivers
  - get_nfvi_type
  - get_stats_acl
  - instantiate
  - scale
  - setup_vnf_environment
  - terminate
  - upload_config_file

- FWApproxVnf
  - collect_kpi
  - deploy_vfw_vnf
  - execute_command
  - get_nfvi_type
  - get_stats_vfw
  - instantiate
  - scale
  - setup_vnf_environment
  - terminate
  - upload_config_file

- VpeApproxVnf
  - collect_kpi
  - execute_command
  - get_stats_vpe
  - instantiate
  - render_template_config
  - scale
  - setup_vnf_environment
  - terminate
  - upload_config_file

- PingTrafficGen
- IxLoadTrafficGen
- ProxApproxVnf
  - get_cpu_id
  - get_cpu_topology
  - instantiate
  - listen_traffic
  - prox_ip_to_hex
  - put_string_to_file
  - rebind_drivers
# Current Traffic Generator Status

<table>
<thead>
<tr>
<th>TG</th>
<th>Priority/Status</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trex</td>
<td></td>
<td>Done – Integrated in Danube</td>
</tr>
<tr>
<td>Ixia/IxNet L1-L4 (using REST API)</td>
<td>Medium</td>
<td>?</td>
</tr>
<tr>
<td>IxLoad L4-L7 (using REST API)</td>
<td>High</td>
<td>In progress, working with REST API</td>
</tr>
<tr>
<td>Spirent</td>
<td>High</td>
<td>?</td>
</tr>
<tr>
<td>Prox</td>
<td>High</td>
<td>Almost done, working on open-sourcing</td>
</tr>
<tr>
<td>Moongen</td>
<td>Low</td>
<td>Not started</td>
</tr>
</tbody>
</table>
KPIs

Using collectd plugins to collect KPIs. Covered by Barometer project

• Plugins
  • Libvirt
  • DPDK Stats
  • DPDK-Events
  • OvS Events
  • Vswitch-stats
  • Huge pages
  • RAM Memory
  • Unreleased plugin in testing to get Memory bandwidth, L1, L2, cache utilization etc.
NetworkService Test Contexts

### Baremetal Linux
- Network application in multiprocess
  - VNF application
  - Control plane / management applications
  - Host OS - User space application
  - NIC
- Eth Driver
- IOMMU
- 1GBe NIC
- L10GE NIC

### Stand-alone Virtualized
- Network application with multiple VMs
  - VNF application
  - Control plane / management applications
  - Socket i/f
  - IOMMU
  - 1GBe NIC
- NIC
- L10GE NIC

### Heat
- Network application with multiple VMs
  - VNF application
  - Control plane / management applications
  - Socket i/f
  - IOMMU
  - 1GBe NIC
- NIC
- L10GE NIC
Summary

• We are extending the Yardstick framework to support testing Network Services
  • Topologies
  • VNFs and VNF descriptors
  • Traffic Generators
  • KPI Collection
We want to hear from you!

Do you have a VNF you want the test?
Do you have a Network Service topology you want to test?

Please come see the Yardstick Team and we can talk.
Backup
PNF with 4 ports

Figure 4.3: Functional architecture for VNF Under Test