VNF Characterization with Yardstick

Ross Brattain
VNF Characterization

Figure 4.3: Functional architecture for VNF Under Test
VNF performance benchmarking
Native Linux environment
Standalone Virtualized environment
Managed virtualized environment (e.g. OpenStack)

Collect KPIs:
Sample Network KPIs, VNF KPIs and NFVi KPIs during the test

Evaluate both scale-up and scale-out performance data
VNFs performance graphs for both scale-up and scale-out in all three environments

Test Infrastructure:
Standard test framework for all 3 environments
Collecting KPIs during the test

• We sample KPIs at multiple intervals to allow for investigation into anomalies during runtime

• Some KPI collection intervals are adjustable

• We collect KPIs from the Traffic generators during the test

• Yardstick collects NFVi KPIs

• There is some reporting in the framework, but really we collect all the KPIs for analytics to process
Architecture of NSB extensions to Yardstick

Note that in some cases mgmt control-plane and data-plane share bridges.

NFVI KPI collection

• Yardstick collects NFVi KPIs
  – Current implementation for Euphrates
    • Requires manual config in pod.yaml
    • Requires SSH access to the compute node(s)
    • Compiles collectd directly on the compute node(s)
    • Plan is to create collectd docker for standardized NFVi KPI collection
      • http://docs.opnfv.org/en/stable-euphrates/submodules/yardstick/docs/testing/user/userguide/13-nsb_operation.html#collectd-kpis
    – Future plan is to use Barometer enabled installers or collectd in docker container.

• We want to be able to correlate NFVI KPIs with individual VNFs
  – If we had core pinning we could match VNF vCPUs to NFVi cores

Dec 2017
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
  – Intel PMU plugin
PMU stats

"data": {
  "compute_0": {
    "core": {
      "cpu": {
        "15": {
          "counter-dTLB-store-misses": "58636",
          "counter-LLC-stores": "489707",
          "counter-alignment-faults": "0",
          "percent-wait": "0",
          "counter-L1-iCache-load-misses": "2939731",
          "counter-minor-faults": "866",
          "percent-softirq": "0.210084033613445",
          "counter-cpu-migrations": "0",
          "counter-emulation-faults": "0",
          "percent-interrupt": "0",
          "counter-cpu-clock": "4917450931",
          "percent-user": "1.68067226890756",
          "counter-branches": "39651833",
          "counter-context-switches": "678",
          "counter-dTLB-load-misses": "306755",
          "counter-cache-references": "5927861",
          "memory_bandwidth-local": "11182080",
          "counter-instructions": "203146046",
          "counter-iTLB-load-misses": "215057",
          "percent-steal": "0",
          "counter-bus-cycles": "25634247",
          "ipc": "0.104876976287796",
          "percent-nice": "0",
          "counter-branch-load-misses": "2467117",
          "counter-task-clock": "4916543925",
          "counter-iTLB-stores": "82729",
          "memory_bandwidth-remote": "2179072",
          "counter-cache-misses": "199009",
          "counter-major-faults": "0",
          "counter-branch-misses": "2415651",
          "counter-L1-dCache-stores": "46462967",
          "counter-branch-loads": "40227471",
          "percent-system": "0.420168067226891",
          "counter-cpu-cycles": "450481129",
          "counter-page-faults": "866",
          "counter-LLC-store-misses": "36495",
          "percent-idle": "97.6890756302521",
          "bytes-llc": "3211264",
          "counter-LLC-load-misses": "40730",
          "counter-LLC-loads": "1097597",
          "counter-L1-dCache-load-misses": "4536261",
          "counter-dTLB-loads": "45401019",
          "counter-L1-dCache-loads": "56572765",
          "counter-dTLB-stores": "43919141"
        }
      }
    }
  }
}

Dec 2017
Different contexts for testing VNFs

Baremetal Linux

Stand-alone Virtualized

Heat

Network application in multiprocess

Network application with multiple VMs

Network application with multiple VMs

VNF application

Control plane / management applications

Host OS

User space application

Control plane / management applications

Host OS – User space application

VNF application

Dpdk stack

pmd

IOMMU

Int Mgmt

NIC 1GbE NIC

Host Kernel with KVM

Int Mgmt

IOMMU

NIC 1GbE NIC

Hardware

Hardware

Hardware

Dec 2017
Heat context changes required for VNF characterization

- Multi-ports per network
  - Match Neutron topology to test topology
  - Specify ports per network, otherwise full mesh by default
- Overriding Heat IP address
  - Some Traffic profiles are static and cannot adjust IP address to make Heat subnet
- Disabling Neutron port security
  - We really want raw L2 access for testing, we don’t want Neutron to filter L2 or L3
- Disabling gateway_ip
  - Otherwise we can’t SSH into VNF due to multiple default routes.
- Disabling DHCP
  - Sometimes we override the Neutron IP addresses with static Traffic Profile IP address, so we don’t need or want DHCP on the port.
- Provider network support
  - SR-IOV
3-Node setup - Correlated Traffic

Dec 2017
Limiting ports per network (3-node correlated traffic)

servers:
  vnf_0:
    floating_ip: true
    placement: "pgrp1"
    network_ports:
      mgmt:
        - mgmt
      uplink_0:
        - xe0
      downlink_0:
        - xe1

tg_0:
  floating_ip: true
  placement: "pgrp1"
  network_ports:
    mgmt:
      - mgmt
    uplink_0:
      - xe0
    downlink_0:
      - xe1

# Trex always needs two ports
uplink_1:
  - xe1
tg_1:
  floating_ip: true
  placement: "pgrp1"
  network_ports:
    mgmt:
      - mgmt
downlink_0:
  - xe0

The traffic generators are connected to a single network only.

VNF is connected to both downlink_0 and uplink_0

networks:
  mgmt:
    cidr: '10.0.1.0/24'
    gateway_ip: 'null'
    port_security_enabled: False
    enable_dhcp: 'false'
  uplink_0:
    cidr: '10.0.2.0/24'
    gateway_ip: 'null'
    port_security_enabled: False
    enable_dhcp: 'false'
downlink_0:
  cidr: '10.0.3.0/24'
  gateway_ip: 'null'
  port_security_enabled: False
  enable_dhcp: 'false'
uplink_1:
  cidr: '10.0.4.0/24'
  gateway_ip: 'null'
  port_security_enabled: False
  enable_dhcp: 'false'
tg_0 and tg_1 are connected to all VNFs

VNFs are only connected to tg_0 and tg_1

Heat correlated scale-4
Overriding Heat allocated IP address

In this case we have a Traffic profile with a fixed src and dest IP so we need to override the IP address provided by Heat.

We can’t just adjust the Heat subnet because we need a specific IP address from the subnet.

Ultimately the Heat subnet is not really necessary if we have raw L2 ports and no port security, we can send spoof any IP address for testing purposes.

https://gerrit.opnfv.org/gerrit/#/c/45393/
SR-IOV Provider networks

Yardstick should support this type of network setup, but we haven’t been able to test yet due to the complexity of deployment.

Yardstick needs help testing this network topology.

You can specify SR-IOV provider networks with the following:

- **networks:**
  - **test-net:**
    - **cidr:** '192.168.1.0/24'
    - **provider:** "sriov"
    - **physical_network:** 'physnet1'

Scale-out

NSB CG-NAT Test (vSwitch) Configuration
Multi-VM – 2x25G per VM, 4 VMs

Scale-out with multiple adapters and VNFs

IXIA*

UDP Replay System
(Swaps IPv4 and MAC Address)
NSB CG-NAT Test (SR-IOV) Configuration
Multi-VM - 2x 10GbE per VM, 10 VMs

Note: Each NIC port is configured with 2 Rx Queues and 3 Tx Queues for Load balancing.
Standardized framework

• Yardstick is designed to support multiple contexts
  – We added k8s context
• Mixed context environment, k8s to VM ping test
• Auto-detection
• Traffic generators work in all contexts
Traffic generator tests

- RFC2544
  - Collecting KPIs during test
- Correlated traffic
  - Using UDP_replay
- RFC 3511
  - Using Ixload
    - Concurrent connections (1B, 4K, 64K, 256K & 1024K transactions)
    - Connections per second (1B, 4K, 64K, 256K & 1024K transactions)
    - HTTP Throughput (1B, 4K, 64K, 256K & 1024K transactions)
    - HTTP Transactions per second (1B, 4K, 64K, 256K & 1024K transactions)
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.

Yardstick doesn’t have auto-discovery of network topologies because we support multiple contexts.

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.
Testcase → Topology → VNF descriptor

Testcase

<table>
<thead>
<tr>
<th>schema:</th>
<th>yardstick:task:0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>scenarios:</td>
<td></td>
</tr>
<tr>
<td>- type:</td>
<td>NSPerf</td>
</tr>
</tbody>
</table>

Topology

| topology: | vpe_vnf_topology.yaml |

VNF descriptor

```yaml
vnfd:vnfd-catalog:
vnfd: vpe_vnf.yaml
  - 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 }}
```

The YAML files are linked by relative pathnames

VNF model:

- member-vnf-index: '1'
  vnfd-id-ref: tg_0
  VNF model: ../../vnfd_descriptors/tg_rfc2544_tpl.yaml

- member-vnf-index: '2'
  vnfd-id-ref: vnf_0
  VNF model: ../../vnfd_descriptors/vpe_vnf.yaml
Sample VNF Descriptor

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.

Password and key_filename must be Python ‘None’ values in order for the Paramiko SSH library to use default SSH auth scheme. Paramiko will check for ssh private keys in the regular locations, then fall back to password auth if provided.
nsd:nsd-catalog:
  nsd:
    - id: 3tg-topology
      name: 3tg-topology
      short-name: 3tg-topology
      description: 3tg-topology
      constituent-vnfd:
        - member-vnf-index: '1'
          vnfd-id-ref: tg__0
          VNF model: ../../vnf_descriptors/tg_rfc2544_tpl.yaml
        - member-vnf-index: '2'
          vnfd-id-ref: vnf__0
          VNF model: ../../vnf_descriptors/acl_vnf.yaml

vld:
  - id: uplink_0
    name: tg__0 to vnf__0 link 1
    type: ELAN
    vnfd-connection-point-ref:
      - member-vnf-index-ref: '1'
        vnfd-connection-point-ref: xe0
        vnfd-id-ref: tg__0
      - member-vnf-index-ref: '2'
        vnfd-connection-point-ref: xe0
        vnfd-id-ref: vnf__0
  - id: downlink_0
    name: vnf__0 to tg__0 link 2
    type: ELAN
    vnfd-connection-point-ref:
      - member-vnf-index-ref: '2'
        vnfd-connection-point-ref: xe1
        vnfd-id-ref: vnf__0
      - member-vnf-index-ref: '1'
        vnfd-connection-point-ref: xe1
        vnfd-id-ref: tg__0

VNF model links to VNF descriptor YAML template

Port pairs are defined as vnfd-connection-point-ref

vld:id maps to Heat Neutron network name

xe0 maps to Heat Neutron port name

Dec 2017
Sample test case

scenarios:
- type: NSPerf
  traffic_profile: ../../traffic_profiles/ipv4_throughput.yaml
  topology: acl-tg-topology.yaml

nodes:
  tg__0: tg_0.yardstick
  vnf__0: vnf_0.yardstick

options:
  framesize:
    uplink: {64B: 100}
    downlink: {64B: 100}

  flow:
    src_ip: [{'tg__0': 'xe0'}]
    dst_ip: [{'tg__0': 'xe1'}]
    count: 1
  traffic_type: 4

  rfc2544:
    allowed_drop_rate: 0.0001 - 0.0001

  vnf__0:
    rules: acl_1rule.yaml

runner:
  type: Iteration
  iterations: 10
  interval: 35

context:
servers:
  vnf__0:
    floating_ip: true
    placement: "pgrp1"
    network_ports:
      mgmt:
        - mgmt
      uplink_0:
        - xe0
      downlink_0:
        - xe1

  tg__0:
    floating_ip: true
    placement: "pgrp1"
    network_ports:
      mgmt:
        - mgmt
      uplink_0:
        - xe0
      downlink_0:
        - xe1

uplink_0 network name and xe0 port name match values in topology

dec 2017

24
Accessing the VNF descriptor from a Python class

```python
global mgmt_interface
global connection

mgmt_interface = self.vnfd_helper.mgmt_interface
self.connection = ssh.SSH.from_node(mgmt_interface)

for port in self.vnfd_helper.port_pairs.all_ports:
    vintf = self.vnfd_helper.find_interface(name=port)['virtual-interface']
    dst_mac = vintf['dst_mac']
    dst_ip = vintf['dst_ip']
    local_mac = vintf['local_mac']
    local_ip = vintf['local_ip']
```

The VNF descriptor is accessed as a Python object using the `self.vnfd_helper` instance variable.

From `vnfd_helper` we can access the external-interfaces list and all the fields, `vpci`, `local_mac`, `dest_mac`, etc.
NetworkServiceTestCase Setup

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)
Sample VNFs

https://wiki.opnfv.org/display/SAM/Technical+Briefs+of+VNFs

- vACL
- vFW
- CGNAPT
How to characterize a new VNF

- Need an image
- Need a flavor
- Create VNF descriptor
- Heat Deploy or pre-Deploy (do you want to use Heat?)
  - Heat Deploy
    - How do we start the VNF after instance is created?
  - Pre-Deploy
    - How do we control the VNF once it is deployed
- Run Test
GenericVNF abstract class

""" Class providing file-like API for generic VNF implementation """

def __init__(self, name, vnfd):
    super(GenericVNF, self).__init__(name, vnfd)
    self.runs_traffic = False

def instantiate(self, scenario_cfg, context_cfg):

def wait_for_instantiate(self):

def terminate(self):

def scale(self, flavor=""):  

def collect_kpi(self):
Controlling the VNF

- You can do anything Python can do
  - REST-API
  - SSH
  - Raw socket
Matching topology to VNF interfaces

- NSB will provide you with a dict listing all the ports created by Heat.
- The ports will have a vld_id that matches the topology.
- uplink_0, downlink_0
- So you may have xe0 on vld_id downlink_0, and xe1 on vld_id uplink_0.
- You need to match the MAC address of xe0 to the interface in the system with that MAC address.
- We usually disable DHCP or gateway_ip on the data-plane interfaces with Heat.
  - Because otherwise we can get multiple gateways or default routes and can’t connect to the mgmt interface on the VNF.
Backup
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
12.8.1.4. SR-IOV 3-Node setup - Correlated Traffic
12.8.1.3. SR-IOV 2-Node setup:
Figure 4.3: Functional architecture for VNF Under Test
The target requirement is to test Network Services consisting of multiple VNFs.