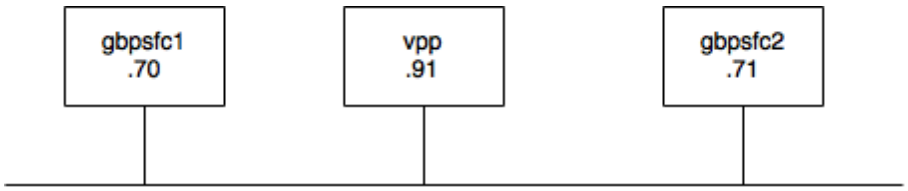
VPP NSH

VXLAN GPE NSH Topology



Terminology:

All service function chaining terminology used within this document is as defined in section 1.4 of RFC 7665

SFF: Service Function Forwarder
SF: Service Function
RSP: Rendered Service Path

Roles:

gbpsfc1: Acts as SFF (OVS) as well as source/destination of packets.
vpp: Acts as SFF with SF (gbpsfc2) attached.
gbpsfc2: Acts as SF.

The RSP is as follows:

gbpsfc1 (sff-source) -> 1,255 -> **vpp** (sff) -> 1,255 -> **gbpsfc2** (SF) -> 1,254 -> **vpp** (sff) -> 1,254 -> **gbpsfc1** (sff-destination)

Initial Setup Instructions:

gbpsfc1 and gbpsfc2 prerequisite setup:

1. Ensure all gbpsfc nodes normally used for other demos are halted/suspended (else you will get IP clash issues).
2. git clone https://git.opendaylight.org/gerrit/p/groupbasedpolicy.git
3. export NUM_NODES=2.
4. vagrant status should only show gbpsfc1 and gbpsfc2.

```
jguichar$ vagrant status
Current machine states:

gbpsfc1                   not created (virtualbox)
gbpsfc2                   not created (virtualbox)
```

5. vagrant up.

Once both gbpsfc nodes are up we need to configure them for their respective desired functionalities.

gbpsfc1:

1. vagrant ssh gbpsfc1.
2. git clone https://git.opendaylight.org/gerrit/sfc
3. cd sfc/sfc-py.
4. Follow instructions for installing sfc client in README.rst "Installation". This is *very* important as there are a number of prerequisites that need to be satisfied and installed on the gbpsfc1 VM.
5. cd sfc (.../sfc/sfc-py/sfc).
6. Make a script called "**genNsh.sh**", and insert following:

```
#!/bin/bash

python3 sff_client.py  --remote-sff-ip=192.168.50.91 --remote-sff-port=4790
--sfp-id=1 --sfp-index=255 --encapsulate=gpe-nsh-ipv4 --inner-src-ip=10.1.1.1
--inner-dest-ip=10.1.1.2 --ctx1=9901 --ctx2=9902 --ctx3=9903 --ctx4=9904
```

7. chmod +x genNsh.sh (no point running yet).

gbpsfc2:

1. Clone following repo: https://github.com/anfredette/vxlan_tool.
2. cd vxlan_tool.
3. Run as per readme found in vxlan_tool directory:

```
sudo python vxlan_tool.py -i 'eth1' -d 'forward' -v 'on'
```

vpp:

1. Make sure you have this patch if not already merged: https://gerrit.fd.io/r/#/c/204/
2. Edit Vagrantfile as follows:

```
cd ... /vpp/build-root/vagrant
export VPP_VAGRANT_NICS=1

edit Vagrantfile
change:
    config.vm.network "private_network", type: "dhcp"
to:
    config.vm.network "private_network", ip: "192.168.50.91"
exit
```

3. vagrant up; vagrant ssh.
4. sudo su.
5. stop vpp.
6. ifconfig eth1 down.
7. ip addr flush dev eth1.
8. start vpp.
9. Create file called /vagrant/nsh.txt:

```
set int state GigabitEthernet0/8/0 up
set int ip address GigabitEthernet0/8/0 192.168.50.91/24
nsh vxlan tunnel src 192.168.50.70 dst 192.168.50.91 c1 9901 c2 9902 c3 9903
c4 9904 spi 1 si 255 vni 16777215 md-type 1 next-ip4 decap-next nsh-vxlan-gpe
nsh vxlan tunnel src 192.168.50.91 dst 192.168.50.71 c1 9901 c2 9902 c3 9903
c4 9904 spi 1 si 255 vni 16777215 md-type 1 next-ip4 decap-next nsh-vxlan-gpe
nsh vxlan tunnel src 192.168.50.71 dst 192.168.50.91 c1 9901 c2 9902 c3 9903
c4 9904 spi 1 si 254 vni 16777215 md-type 1 next-ip4 decap-next nsh-vxlan-gpe
nsh vxlan tunnel src 192.168.50.91 dst 192.168.50.70 c1 9901 c2 9902 c3 9903
c4 9904 spi 1 si 254 vni 16777215 md-type 1 next-ip4 decap-next nsh-vxlan-gpe
```

10. vppctl show interface (should see GigabitEthernet and 4 tunnels).

```
root@localhost:/home/vagrant# vppctl show interface
              Name               Idx       State          Counter
Count
GigabitEthernet0/8/0               5         up
nsh_vxlan_gpe_tunnel0              6         up
nsh_vxlan_gpe_tunnel1              7         up
nsh_vxlan_gpe_tunnel2              8         up
nsh_vxlan_gpe_tunnel3              9         up
```

11. vppctl ex /vagrant/nsh.txt.
12. Configuration of the necessary NSH vxlan tunnels should have been executed.

```
vpp# show nsh vxlan tunnel
[0] 192.168.50.70 (src) 192.168.50.91 (dst) fibs: encap 0, decap 0 decap next
nsh-vxlan-gpe
  vxlan VNI 16777215 nsh ver 0 len 6 (24 bytes) md_type 1 next_protocol 1
  service path 1 service index 255
  c1 9901 c2 9902 c3 9903 c4 9904
[1] 192.168.50.91 (src) 192.168.50.71 (dst) fibs: encap 0, decap 0 decap next
nsh-vxlan-gpe
  vxlan VNI 16777215 nsh ver 0 len 6 (24 bytes) md_type 1 next_protocol 1
  service path 1 service index 255
  c1 9901 c2 9902 c3 9903 c4 9904
[2] 192.168.50.71 (src) 192.168.50.91 (dst) fibs: encap 0, decap 0 decap next
nsh-vxlan-gpe
  vxlan VNI 16777215 nsh ver 0 len 6 (24 bytes) md_type 1 next_protocol 1
  service path 1 service index 254
  c1 9901 c2 9902 c3 9903 c4 9904
[3] 192.168.50.91 (src) 192.168.50.70 (dst) fibs: encap 0, decap 0 decap next
nsh-vxlan-gpe
  vxlan VNI 16777215 nsh ver 0 len 6 (24 bytes) md_type 1 next_protocol 1
  service path 1 service index 254
  c1 9901 c2 9902 c3 9903 c4 9904
```

13. Set trace so can capture the VPP code path:

```
root@localhost:/home/vagrant# vppctl
vpp# trace add dpdk-input 10
```

Testing:

Basic test shows packets sent from gbpsfc1 with SFP-id 1, SI 255. Packets are forwarded by VPP to gbpsfc2 that returns the packets with SI decremented by 1. VPP forwards packets back to gbpsfc1.

gbpsfc1:

1. Generate packets using the genNSH script

```
vagrant@gbpsfc1:~/sfc/sfc-py/sfc$ sudo su
root@gbpsfc1:/home/vagrant/sfc/sfc-py/sfc# ./genNSH.sh
```

```
INFO:__main__:Sending VXLAN-GPE/NSH/IPv4 packet to SFF: ('192.168.50.91',
4790)
INFO:__main__:Received packet from SFF: ('192.168.50.91', 4790)
```

2. May need to run ./genNSH.sh couple times for ARP to resolve.
3. Full packet capture at gbpsfc1 can be collected using **tcpdump -n -i eth1**

vpp:

1. Check errors to see how incoming packets from gbpsfc1 are being handled:

```
vpp# show error
      Count                    Node                  Reason
            2          nsh-vxlan-gpe-input        good packets decapsulated
            2          nsh-vxlan-gpe-encap        good packets encapsulated
            2               arp-input             ARP replies sent
```

2. 'good packets decapsulated' shows that VPP successfully received packets from gbpsfc1 and decapsulated. 'good packets encapsulated' shows that VPP successfully encapsulated and forwarded based on the NSH SPI/SI combination to gbpsfc2.
3. We can verify the processing by looking at the trace log.

```
vpp# show trace
------------------- Start of thread 0 vpp_main -------------------
Packet 1

02:22:11:181830: dpdk-input
  GigabitEthernet0/8/0 rx queue 0
  buffer 0xa737: current data 0, length 106, free-list 0, totlen-nifb 0, trace
0x0
  PKT MBUF: port 0, nb_segs 1, pkt_len 106
    buf_len 2304, data_len 106, ol_flags 0x0,
    packet_type 0x0
  IP4: 08:00:27:ae:69:22 -> 08:00:27:f7:0c:69
  UDP: 192.168.50.70 -> 192.168.50.91 // from gbpsfc1 -> vpp
    tos 0x00, ttl 64, length 92, checksum 0x4681
    fragment id 0x0e1e, flags DONT_FRAGMENT
  UDP: 4790 -> 4790 // its vxlan-gpe
    length 72, checksum 0x615c
02:22:11:181892: ethernet-input
  IP4: 08:00:27:ae:69:22 -> 08:00:27:f7:0c:69
02:22:11:181903: ip4-input
  UDP: 192.168.50.70 -> 192.168.50.91
    tos 0x00, ttl 64, length 92, checksum 0x4681
    fragment id 0x0e1e, flags DONT_FRAGMENT
```

```
  UDP: 4790 -> 4790
    length 72, checksum 0x615c
02:22:11:181915: ip4-local
  fib 0 adj-idx 4 : local 192.168.50.91/24 flow hash: 0x00000000
02:22:11:181920: ip4-udp-lookup
  UDP: src-port 4790 dst-port 4790
02:22:11:181927: nsh-vxlan-gpe-input
  NSH-VXLAN: tunnel 0 next 4 error 0 // entering on tunnel [0]
  ver 0 len 6 (24 bytes) md_type 1 next_protocol 1
  spi 1 si 255 c1 9901 c2 9902 c3 9903 c4 9904 // incoming NSH SPI, SI
02:22:11:181932: nsh-vxlan-gpe-encap
  NSH-VXLAN-ENCAP: tunnel 1 // being encapsulated into tunnel [1]
02:22:11:181938: ip4-rewrite-transit
  fib 0 adj-idx 6 : GigabitEthernet0/8/0
                    IP4: 08:00:27:f7:0c:69 -> 08:00:27:7d:48:b6 flow hash:
0x00000000
  IP4: 08:00:27:f7:0c:69 -> 08:00:27:7d:48:b6
  UDP: 192.168.50.91 -> 192.168.50.71
    tos 0x00, ttl 253, length 92, checksum 0xd79d
    fragment id 0x0000
  UDP: 4790 -> 4790
    length 72, checksum 0x0000
02:22:11:181941: GigabitEthernet0/8/0-output
  GigabitEthernet0/8/0
  IP4: 08:00:27:f7:0c:69 -> 08:00:27:7d:48:b6
  UDP: 192.168.50.91 -> 192.168.50.71
    tos 0x00, ttl 253, length 92, checksum 0xd79d
    fragment id 0x0000
  UDP: 4790 -> 4790
    length 72, checksum 0x0000
02:22:11:181945: GigabitEthernet0/8/0-tx
  GigabitEthernet0/8/0 tx queue 0
  buffer 0xa737: current data 0, length 106, free-list 0, totlen-nifb 0, trace
0x0
  IP4: 08:00:27:f7:0c:69 -> 08:00:27:7d:48:b6
  UDP: 192.168.50.91 -> 192.168.50.71 // from vpp to gbpsfc2
    tos 0x00, ttl 253, length 92, checksum 0xd79d
    fragment id 0x0000
  UDP: 4790 -> 4790
    length 72, checksum 0x0000
```

4. The above trace shows the packet arriving at VPP and being successfully forwarded to gbpsfc2 through interface GigabitEthernet0/8/0.

```
vpp# show trace
------------------- Start of thread 0 vpp_main -------------------
Packet 3

02:22:11:182530: dpdk-input
  GigabitEthernet0/8/0 rx queue 0
  buffer 0xa6e9: current data 0, length 106, free-list 0, totlen-nifb 0, trace
0x2
  PKT MBUF: port 0, nb_segs 1, pkt_len 106
    buf_len 2304, data_len 106, ol_flags 0x0,
    packet_type 0x0
  IP4: 08:00:27:7d:48:b6 -> 08:00:27:f7:0c:69
  UDP: 192.168.50.71 -> 192.168.50.91 // from gbpsfc2 to VPP
```

```
    tos 0x00, ttl 255, length 92, checksum 0x016c
    fragment id 0xd431
  UDP: 4790 -> 4790 // its vxlan-gpe
    length 72, checksum 0x615c
02:22:11:182549: ethernet-input
  IP4: 08:00:27:7d:48:b6 -> 08:00:27:f7:0c:69
02:22:11:182553: ip4-input
  UDP: 192.168.50.71 -> 192.168.50.91
    tos 0x00, ttl 255, length 92, checksum 0x016c
    fragment id 0xd431
  UDP: 4790 -> 4790
    length 72, checksum 0x615c
02:22:11:182555: ip4-local
  fib 0 adj-idx 4 : local 192.168.50.91/24 flow hash: 0x00000000
02:22:11:182558: ip4-udp-lookup
  UDP: src-port 4790 dst-port 4790
02:22:11:182560: nsh-vxlan-gpe-input
  NSH-VXLAN: tunnel 2 next 4 error 0 // packets arriving on tunnel [2]
  ver 0 len 6 (24 bytes) md_type 1 next_protocol 1
  spi 1 si 254 c1 9901 c2 9902 c3 9903 c4 9904 // SI decremented by 1
02:22:11:182564: nsh-vxlan-gpe-encap
  NSH-VXLAN-ENCAP: tunnel 3 // being encapsulated into tunnel [3]
02:22:11:182567: ip4-rewrite-transit
  fib 0 adj-idx 8 : GigabitEthernet0/8/0
                    IP4: 08:00:27:f7:0c:69 -> 08:00:27:ae:69:22 flow hash:
0x00000000
  IP4: 08:00:27:f7:0c:69 -> 08:00:27:ae:69:22
  UDP: 192.168.50.91 -> 192.168.50.70
    tos 0x00, ttl 253, length 92, checksum 0xd79e
    fragment id 0x0000
  UDP: 4790 -> 4790
    length 72, checksum 0x0000
02:22:11:182569: GigabitEthernet0/8/0-output
  GigabitEthernet0/8/0
  IP4: 08:00:27:f7:0c:69 -> 08:00:27:ae:69:22
  UDP: 192.168.50.91 -> 192.168.50.70
    tos 0x00, ttl 253, length 92, checksum 0xd79e
    fragment id 0x0000
  UDP: 4790 -> 4790
    length 72, checksum 0x0000
02:22:11:182572: GigabitEthernet0/8/0-tx
  GigabitEthernet0/8/0 tx queue 0
  buffer 0xa6e9: current data 0, length 106, free-list 0, totlen-nifb 0, trace
0x2
  IP4: 08:00:27:f7:0c:69 -> 08:00:27:ae:69:22
  UDP: 192.168.50.91 -> 192.168.50.70 // from vpp back to gbpsfc1
    tos 0x00, ttl 253, length 92, checksum 0xd79e
    fragment id 0x0000
  UDP: 4790 -> 4790
    length 72, checksum 0x0000
```

5. The above trace output shows the packet coming back from gbpsfc2 and being forwarded back to gbpsfc1 successfully.

gbpsfc2:

1. Packet receipt can be reviewed at gbpsfc2

```
vagrant@gbpsfc2:~/vxlan_tool$ sudo python vxlan_tool.py -i 'eth1' -d 'forward'
-v 'on'

Packet #1
Eth Dst MAC: 08:00:27:7d:48:b6, Src MAC: 08:00:27:f7:0c:69, Ethertype: 0x0800
IP Version: 4 IP Header Length: 5, TTL: 253, Protocol: 17, Src IP:
192.168.50.91, Dst IP: 192.168.50.71
UDP Src Port: 4790, Dst Port: 4790, Length: 72, Checksum: 0
VxLAN/VxLAN-gpe VNI: 16777215, flags: 0c, Next: 4
NSH base nsp: 1, nsi: 255
NSH context c1: 0x000026ad, c2: 0x000026ae, c3: 0x000026af, c4: 0x000026b0
```