vCPE

Test Plan

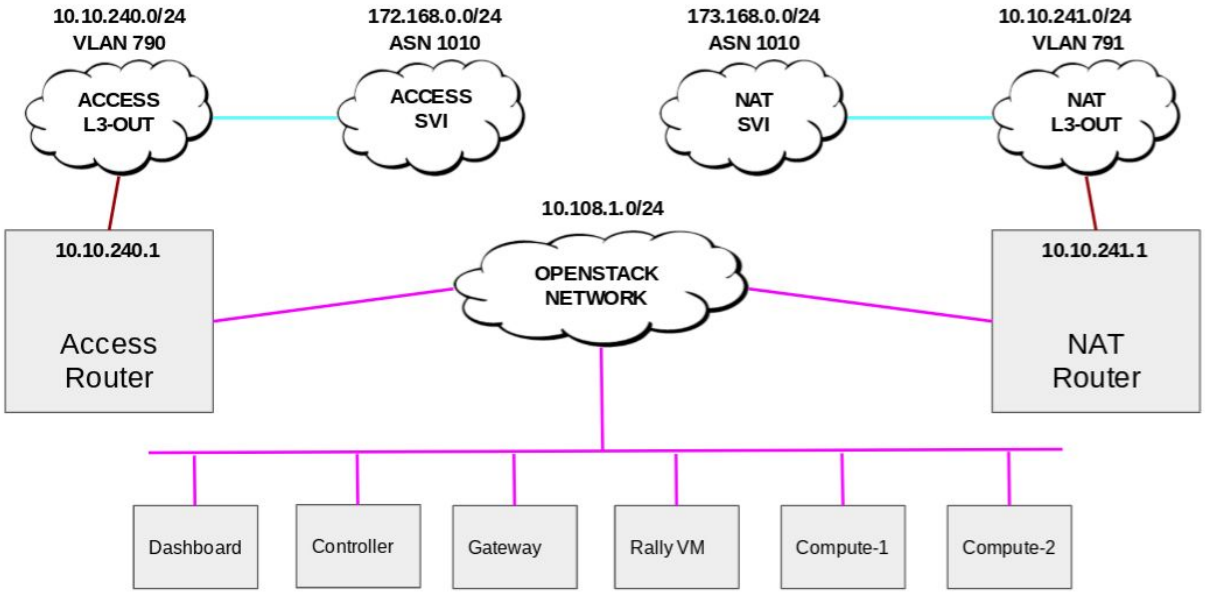
vCPE Topology

This section covers the top level infrastructure topology used for vCPE solution testing. All of the test cases use part or whole of this topology based on the use case. In this use case we cover basic sanity and reachability of networks from the VM from where the script execution will take place.

Setup Details

The below picture depicts the framework of the setup used for vCPE solution testing. The openstack installation is outside of the vCPE solution. The openstack components and compute nodes are accessible on the openstack network. The rally VM is where the scripts are hosted. Access and NAT routers are used to simulate the customer side and internet side of the vCPE solution. Access and NAT SVI's are created to have common access to multiple customers vCPE deployments. The Access SVI is linked to Access L3-OUT and the NAT SVI is linked to NAT L3-OUT for reachability to networks outside of Cisco ACI. The Access and NAT SVIs are used across all the vCPE use cases. The L3-OUT VRFs are preconfigured with static routes for reachability to the access and nat routers as well as the rally VM.

Setup Diagram



ACI Configuration

Below is the snippet of the ACI configuration from APIC for L3-Outs to reach the external networks used in vCPE testing.

```
 leaf 101
   vrf context tenant common vrf access-out-vrf l3out Access-Out
     router-id 199.199.199.0
     ip route 10.10.224.0/20 10.10.240.1 1
     ip route 10.10.231.0/24 10.10.240.1 1
     ip route 10.108.1.0/24 10.10.240.1 1
     exit

   vrf context tenant common vrf internet-out-vrf l3out Internet-Out
     router-id 199.199.199.0
     ip route 10.108.1.0/24 10.10.241.1 1
     ip route 8.8.8.0/24 10.10.241.1 1
     exit
   interface vlan 790
     vrf member tenant common vrf access-out-vrf
     ip address 10.10.240.2/24
     exit
   interface vlan 791
     vrf member tenant common vrf internet-out-vrf
     ip address 10.10.241.2/24
     exit

   interface ethernet 1/21
     # policy-group OC-LAP
     switchport trunk allowed vlan 790 tenant common external-svi l3out Access-Out
     switchport trunk allowed vlan 791 tenant common external-svi l3out Internet-Out
     exit
```

Functional Test Cases

Reachability check

In this test scenario, we create Access SVI and Internet/NAT SVI. These SVIs are left as is as they work as a base framework to run the rest of the test cases. The reachability of SVI IP address is tested from the Rally VM where the scripts are hosted and from where the test scripts will be run.

Test Steps	**Test Results**
Create access network with following attributes --provider:network_type vlan --shared --apic:svi True --apic:bgp_enable True --apic:bgp_asn 1010 --apic:distinguished_names type=dict ExternalNetwork=uni/tn-comment/out-Access-Out/instP-data_ext_pol	The network should be successfully created without any errors
Create subnet for access network with following attributes --gateway 172.168.0.1 --subnet-range 172.168.0.0/24 --host-route destination=10.108.1.0/24,gateway=172.168.0.1 --host-route destination=10.10.240.0/24,gateway=172.168.0.1 --host-route destination=10.10.224.0/22,gateway=172.168.0.1	The subnet should be created successfully without any errors
Create SVI ports on all leaf switches with the following attributes. Change the IP address for each node. --device-owner apic:svi --fixed-ip subnet=”subnet-id”,ip-address=172.168.0.199	The SVI ports for all the leaf switches should be created successfully without any errors
Ping default gateway from access router	Ping should be successful
Ping SVI port IP address from access router	Ping should be successful
Ping default gateway from rally vm	Ping should be successful
Ping SVI port IP address from rally vm	Ping should be successful

Create internet network with following attributes --provider:network_type vlan --shared --apic:svi True --apic:bgp_enable True --apic:bgp_asn 1020 --apic:distinguished_names type=dict ExternalNetwork=uni/tn-common/out-Internet-Out/instP-data_ext_pol	The network should be successfully created without any errors
Create subnet for access network with following attributes --gateway 173.168.0.1 --subnet-range 173.168.0.0/24 --host-route destination=10.108.1.0/24,gateway=173.168.0.1 --host-route destination=10.10.241.0/24,gateway=173.168.0.1	The subnet should be created successfully without any errors
Create SVI ports on all leaf switches with the following attributes. Change the IP address for each node. --device-owner apic:svi --fixed-ip subnet=”subnet-id”,ip-address=173.168.0.199	The SVI ports for all the leaf switches should be created successfully without any errors
Ping default gateway from nat router	Ping should be successful
Ping SVI port IP address from nat router	Ping should be successful
Ping default gateway from rally vm	Ping should be successful
Ping SVI port IP address from rally vm	Ping should be successful

Service Function Chaining

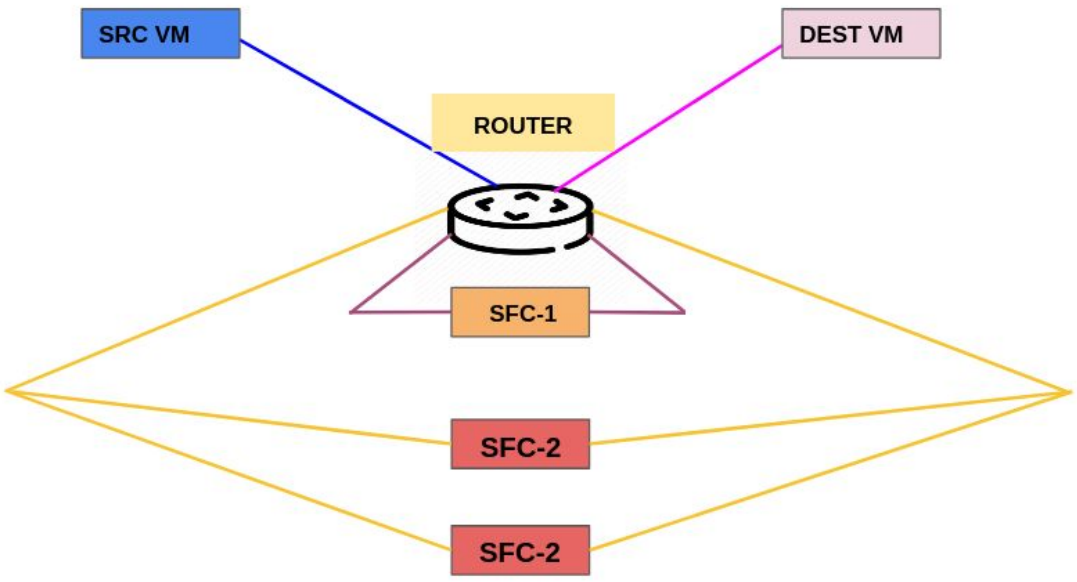
Cisco Application Centric Infrastructure (ACI) technology provides the capability to insert Layer 4 through Layer 7 functions using an approach called a service graph. The industry normally refers to the capability to add Layer 4 through Layer 7 devices in the path between endpoints as service insertion. The Cisco ACI service graph technology can be considered a superset of service insertion.

Cisco Openstack Neutron Plugin for ACI provides the capability to create a multi-node service graph using neutron SFC API. This allows us to create service chains straight from Openstack without doing any configuration on ACI.

Setup Details

The setup for testing Cisco ACI service graph using openstack neutron service function chaining constitutes of a source network and a destination network. Along with this based on the use case, multiple other networks will be created to host the SFC vms. Also multiple VMs can be spun on source and destination networks for traffic validation. All the networks used in the functional testing will be connected to a Router.

Setup Diagram



Functional Test Cases

Base Topology and Configuration

Steps to build base Topology

Test Steps	**Test Results**
Create an openstack router	Openstack router created successfully.
Create source network with following attributes	Source network created successfully
Create source subnet with following attributes --gateway 192.168.10.1 --subnet-range 192.168.10.0/24	Source subnet created successfully
Add the source subnet to the openstack router	Subnet added successfully
Create a port on the access network	Port created successfully
Create a port on the source network	Port created successfully
Create Source VM with ports from access network and source network	Source VM created successfully
Ping the gateway address from Source VM	Ping successful
Create destination network with following attributes	Destination network created successfully
Create destination subnet with following attributes --gateway 192.168.20.1 --subnet-range 192.168.20.0/24	Destination subnet created successfully
Add the destination subnet to the openstack router	Subnet added successfully
Create a port on the access network	Port created successfully
Create a port on the destination network	Port created successfully
Create a VM with ports from access network and destination network	Destination VM created successfully
Ping the gateway address from Destination VM	Ping successful
Ping multiple IP address on Destination VM from	Ping successful

the Source VM (8.8.8.1-8) range	

Single Service Function

This use case will have a single service function and it is for basic functionality of SFC. Traffic will go from source VM to destination VM through the SFC VM.

Test Steps	**Test Results**
Create LEFT-1 network	LEFT-1 network created successfully
Create LEFT-1 subnet with following attributes --gateway 1.1.0.1 --subnet-range 1.1.0.0/24 --host-route destination=192.168.10.0/24,gateway=1.1.0.1	LEFT-1 subnet created successfully
Add LEFT-1 subnet to the openstack router	Subnet added successfully
Create RIGHT-1 network	RIGHT-1 network created successfully
Create RIGHT-1 subnet with following attributes --gateway 2.1.0.1 --subnet-range 2.1.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.1.0.1 --host-route destination=128.0.0.0/1,gateway=2.1.0.1	RIGHT-1 subnet created successfully
Add RIGHT-1 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-1 network	Port created successfully
Create a port on RIGHT-1 network	Port created successfully
Create a Service VM with ports from LEFT-1 and RIGHT-1 networks with OpenWRTSI1 image	Service VM created successfully
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 192.168.10.0/24 --l7-parameters logical_source_network=SRC-NET-ID, logical_destination_network=DEST-NET-ID	Flow Classifier created successfully
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully

Ping the Destination VM from the Source VM	Ping successful and still going through the router
Create a port chain with the flow classifier and port pair group	Service Function Chain created successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	Ping successful and going through the newly created SFC except 8.8.8.1

Multiple instances of the same Service Function

This use case will cover the same service function instantiated multiple times for load balancing. The traffic from source VM will go through one of the service VMs based on the hash logic to get to the destination VM.

Test Steps	**Test Results**
Create LEFT-1 network	LEFT-1 network created successfully
Create LEFT-1 subnet with following attributes --gateway 1.1.0.1 --subnet-range 1.1.0.0/24 --host-route destination=192.168.10.0/24,gateway=1.1.0.1	LEFT-1 subnet created successfully
Add LEFT-1 subnet to the openstack router	Subnet added successfully
Create RIGHT-1 network	RIGHT-1 network created successfully
Create RIGHT-1 subnet with following attributes --gateway 2.1.0.1 --subnet-range 2.1.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.1.0.1 --host-route destination=128.0.0.0/1,gateway=2.1.0.1	RIGHT-1 subnet created successfully
Add RIGHT-1 subnet to the openstack router	Subnet added successfully
Create two ports on LEFT-1 network	Ports created successfully
Create two ports on RIGHT-1 network	Ports created successfully
Create two Service VMs with ports from LEFT-1 and RIGHT-1 networks with OpenWRTSI1 image	Service VMs created successfully
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 192.168.10.0/24	Flow Classifier created successfully

--l7-parameters logical_source_network=SRC-NET-ID, logical_destination_network=DEST-NET-ID	
Create two port pair with ports used for creating the service VM-1 and service VM-2	Port Pairs create successfully
Create a port pair group with the port pairs created	Port Pair Group create successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.x) range	Ping successful for all IP address
Create a port chain with the flow classifier and port pair group	Service Function Chain created successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	Ping successful and going through different newly created SFCs based on hash except for 8.8.8.1 which is blocked

Multiple Service Functions in a Chain

This use case will cover multiple service functions in a chain. Traffic from source VM has to traverse the entire chain to get to destination VM

Test Steps	**Test Results**
Create LEFT-1 network	LEFT-1 network created successfully
Create LEFT-1 subnet with following attributes --gateway 1.1.0.1 --subnet-range 1.1.0.0/24 --host-route destination=192.168.10.0/24,gateway=1.1.0.1	LEFT-1 subnet created successfully
Add LEFT-1 subnet to the openstack router	Subnet added successfully
Create RIGHT-1 network	RIGHT-1 network created successfully
Create RIGHT-1 subnet with following attributes --gateway 2.1.0.1 --subnet-range 2.1.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.2.0.1 --host-route destination=128.0.0.0/1,gateway=2.2.0.1	RIGHT-1 subnet created successfully
Add RIGHT-1 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-1 network	Port created successfully

Create a port on RIGHT-1 network	Port created successfully
Create a Service VM with ports from LEFT-1 and RIGHT-1 networks with OpenWRTSI1 image	Service VM created successfully
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 192.168.10.0/24 --l7-parameters logical_source_network=SRC-NET-ID, logical_destination_network=DEST-NET-ID	Flow Classifier created successfully
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully
Create LEFT-2 network	LEFT-2 network created successfully
Create LEFT-2 subnet with following attributes --gateway 1.2.0.1 --subnet-range 1.2.0.0/24 --host-route destination=192.168.10.0/24,gateway=1.1.0.1	LEFT-2 subnet created successfully
Add LEFT-2 subnet to the openstack router	Subnet added successfully
Create RIGHT-2 network	RIGHT-2 network created successfully
Create RIGHT-2 subnet with following attributes --gateway 2.2.0.1 --subnet-range 2.2.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.2.0.1 --host-route destination=128.0.0.0/1,gateway=2.2.0.1	RIGHT-2 subnet created successfully
Add RIGHT-2 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-2 network	Port created successfully
Create a port on RIGHT-2 network	Port created successfully
Create a Service VM with ports from LEFT-2 and RIGHT-2 networks with OpenWRTSI2 image	Service VM created successfully
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully

Ping the Destination VM from the Source VM	All Ping successful as they are going through the router
Create a port chain with the flow classifier and port pair group	Service Function Chain created successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	All Ping successful except 8.8.8.1, 8.8.8.2. Blocked by Service VMs

Update the SFC by adding new port-pair to the group for scale up

In this use case we will spin a new service VM, create a new port-pair and update the existing port-pair-group with a new port-pair and check if traffic will traverse the new service VM.

Test Steps	**Test Results**
Create LEFT-1 network	LEFT-1 network created successfully
Create LEFT-1 subnet with following attributes --gateway 1.1.0.1 --subnet-range 1.1.0.0/24 --host-route destination=192.168.10.0/24,gateway=1.1.0.1	LEFT-1 subnet created successfully
Add LEFT-1 subnet to the openstack router	Subnet added successfully
Create RIGHT-1 network	RIGHT-1 network created successfully
Create RIGHT-1 subnet with following attributes --gateway 2.2.0.1 --subnet-range 2.2.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.2.0.1 --host-route destination=128.0.0.0/1,gateway=2.2.0.1	RIGHT-1 subnet created successfully
Add RIGHT-1 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-1 network	Port created successfully
Create a port on RIGHT-1 network	Port created successfully
Create a Service VM with ports from LEFT-1 and RIGHT-1 networks with OpenWRTSI1 image	Service VM created successfully
Create a flow classifier with following attributes	Flow Classifier created successfully

--destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 192.168.10.0/24 --l7-parameters logical_source_network=SRC-NET-ID, logical_destination_network=DEST-NET-ID	
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully
Ping the Destination VM from the Source VM	Ping successful and still going through the router
Create a port chain with the flow classifier and port pair group	Service Function Chain created successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	All Ping successful except 8.8.8.1
Create a new port on LEFT-1 network	Port created successfully
Create a new port on RIGHT-1 network	Port created successfully
Create a Service VM with new ports from LEFT-1 and RIGHT-1 networks with OpenWRTSI1 image	Service VM created successfully
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Update the existing port pair group with the newly created port pair	Port Pair Group updated successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	All Ping successful except 8.8.8.1. Also based on hash traffic will go through both the service VMs

Update the SFC by adding new flow classifier

In this use case we will update an existing service chain with a new flow classifier and check if traffic is impacted for the hosts in the new flow classifier.

Test Steps	**Test Results**
Create LEFT-1 network	LEFT-1 network created successfully

Create LEFT-1 subnet with following attributes --gateway 1.1.0.1 --subnet-range 1.1.0.0/24 --host-route destination=192.168.10.0/24,gateway=1.1.0.1	LEFT-1 subnet created successfully
Add LEFT-1 subnet to the openstack router	Subnet added successfully
Create RIGHT-1 network	RIGHT-1 network created successfully
Create RIGHT-1 subnet with following attributes --gateway 2.2.0.1 --subnet-range 2.2.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.2.0.1 --host-route destination=128.0.0.0/1,gateway=2.2.0.1	RIGHT-1 subnet created successfully
Add RIGHT-1 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-1 network	Port created successfully
Create a port on RIGHT-1 network	Port created successfully
Create a Service VM with ports from LEFT-1 and RIGHT-1 networks with OpenWRTSI1 image	Service VM created successfully
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 192.168.10.101/32 --l7-parameters logical_source_network=SRC-NET-ID, logical_destination_network=DEST-NET-ID	Flow Classifier created successfully
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully
Ping the Destination VM from the Source VM	Ping successful and still going through the router
Create a port chain with the flow classifier and port pair group	Service Function Chain created successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	All Ping successful except 8.8.8.1
Create a new port on the access network	Port created successfully

Create a new port on the source network with specific IP address 192.168.10.102	Port created successfully
Create Source VM with new ports from access network and source network	Source VM created successfully
Ping the gateway address from new Source VM	Ping successful
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 192.168.10.102/32 --l7-parameters logical_source_network=SRC-NET-ID, logical_destination_network=DEST-NET-ID	Flow Classifier created successfully
Update the existing port chain with newly created flow classifier	Service Function Chain updated successfully
Ping multiple IP address on Destination VM from the Source VM-1 (8.8.8.1-8) range	All Ping successful except 8.8.8.1 Blocked by Service VMs
Ping multiple IP address on Destination VM from the Source VM-2 (8.8.8.1-8) range	All Ping successful except 8.8.8.1 Blocked by Service VMs

Update the SFC by deleting one of the port-pair from the group for scale down

In this use case we will delete a port-pair from an existing port-pair-group and check if traffic will fall back to other existing service VMs

Test Steps	**Test Results**
Create LEFT-1 network	LEFT-1 network created successfully
Create LEFT-1 subnet with following attributes --gateway 1.1.0.1 --subnet-range 1.1.0.0/24 --host-route destination=192.168.10.0/24,gateway=1.1.0.1	LEFT-1 subnet created successfully
Add LEFT-1 subnet to the openstack router	Subnet added successfully
Create RIGHT-1 network	RIGHT-1 network created successfully
Create RIGHT-1 subnet with following attributes	RIGHT-1 subnet created successfully

--gateway 2.2.0.1 --subnet-range 2.2.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.2.0.1 --host-route destination=128.0.0.0/1,gateway=2.2.0.1	
Add RIGHT-1 subnet to the openstack router	Subnet added successfully
Create two ports on LEFT-1 network	Ports created successfully
Create two ports on RIGHT-1 network	Ports created successfully
Create two Service VM with ports from LEFT-1 and RIGHT-1 networks with OpenWRTSI1 image	Service VMs created successfully
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 192.168.10.0/24 --l7-parameters logical_source_network=SRC-NET-ID, logical_destination_network=DEST-NET-ID	Flow Classifier created successfully
Create two port pairs with ports used for creating the service VMs	Port Pairs create successfully
Create a port pair group with the port pairs created	Port Pair Group create successfully
Ping the Destination VM from the Source VM	Ping successful and still going through the router
Create a port chain with the flow classifier and port pair group	Service Function Chain created successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	All Ping successful except 8.8.8.1 and base on hash traffic flows through both service VMs
Update the port pair group by removing a port pair out of the two port pairs we have	Port pair group updated successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	All Ping successful except 8.8.8.1 Blocked by Service VMs. All the traffic should go through service VM one only as this is the only active VM in the service function chain.

Update the SFC by deleting an existing flow classifier

In this use case we will update an existing service chain by deleting a flow classifier and check if traffic is rerouted to existing service function VMs.

Test Steps	**Test Results**
Create LEFT-1 network	LEFT-1 network created successfully
Create LEFT-1 subnet with following attributes --gateway 1.1.0.1 --subnet-range 1.1.0.0/24 --host-route destination=192.168.10.0/24,gateway=1.1.0.1	LEFT-1 subnet created successfully
Add LEFT-1 subnet to the openstack router	Subnet added successfully
Create RIGHT-1 network	RIGHT-1 network created successfully
Create RIGHT-1 subnet with following attributes --gateway 2.2.0.1 --subnet-range 2.2.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.2.0.1 --host-route destination=128.0.0.0/1,gateway=2.2.0.1	RIGHT-1 subnet created successfully
Add RIGHT-1 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-1 network	Port created successfully
Create a port on RIGHT-1 network	Port created successfully
Create a Service VM with ports from LEFT-1 and RIGHT-1 networks with OpenWRTSI1 image	Service VM created successfully
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 192.168.10.101/32 --l7-parameters logical_source_network=SRC-NET-ID, logical_destination_network=DEST-NET-ID	Flow Classifier created successfully
Create a second flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 192.168.10.102/32	Flow Classifier created successfully

--l7-parameters logical_source_network=SRC-NET-ID, logical_destination_network=DEST-NET-ID	
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully
Ping the Destination VM from the Source VMs	Ping successful and still going through the router
Create a port chain with both the flow classifier and port pair group	Service Function Chain created successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range from both source VMs	All Ping successful except 8.8.8.1 as it is blocked by Service VM
Update the existing port chain where we remove the flow classifier with IP source 192.168.10.102	Service Function Chain updated successfully
Ping multiple IP address on Destination VM from the Source VM-1 (8.8.8.1-8) range	All Ping successful except 8.8.8.1 Blocked by Service VMs
Ping multiple IP address on Destination VM from the Source VM-2 (8.8.8.1-8) range	All Ping successful as they are going directly from the router as service function chain is not in effect.

Add new service to an existing SFC

In this use case we will start off with a single service in SFC and while the traffic is going, add one more service to the SFC and check if traffic is going thru the newly added service

Test Steps	**Test Results**
Create LEFT-1 network	LEFT-1 network created successfully
Create LEFT-1 subnet with following attributes --gateway 1.1.0.1 --subnet-range 1.1.0.0/24 --host-route destination=192.168.10.0/24,gateway=1.1.0.1	LEFT-1 subnet created successfully
Add LEFT-1 subnet to the openstack router	Subnet added successfully

Create RIGHT-1 network	RIGHT-1 network created successfully
Create RIGHT-1 subnet with following attributes --gateway 2.2.0.1 --subnet-range 2.2.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.2.0.1 --host-route destination=128.0.0.0/1,gateway=2.2.0.1	RIGHT-1 subnet created successfully
Add RIGHT-1 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-1 network	Port created successfully
Create a port on RIGHT-1 network	Port created successfully
Create a Service VM with ports from LEFT-1 and RIGHT-1 networks with OpenWRTSI1 image	Service VM created successfully
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 192.168.10.0/24 --l7-parameters logical_source_network=SRC-NET-ID, logical_destination_network=DEST-NET-ID	Flow Classifier created successfully
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully
Ping the Destination VM from the Source VM	Ping successful and still going through the router
Create a port chain with the flow classifier and port pair group	Service Function Chain created successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	All Ping successful except 8.8.8.1
Create LEFT-2 network	LEFT-2 network created successfully
Create LEFT-2 subnet with following attributes --gateway 1.2.0.1 --subnet-range 1.2.0.0/24 --host-route destination=192.168.10.0/24,gateway=1.1.0.1	LEFT-2 subnet created successfully
Add LEFT-2 subnet to the openstack router	Subnet added successfully

Create RIGHT-2 network	RIGHT-2 network created successfully
Create RIGHT-2 subnet with following attributes --gateway 2.2.0.1 --subnet-range 2.2.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.2.0.1 --host-route destination=128.0.0.0/1,gateway=2.2.0.1	RIGHT-2 subnet created successfully
Add RIGHT-2 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-2 network	Port created successfully
Create a port on RIGHT-2 network	Port created successfully
Create a Service VM with ports from LEFT-2 and RIGHT-2 networks with OpenWRTSI2 image	Service VM created successfully
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully
Ping the Destination VM from the Source VM	All Ping successful except 8.8.8.1
Update the existing port chain with newly created port pair group	Service Function Chain updated successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	All Ping successful except 8.8.8.1, 8.8.8.2. Blocked by Service VMs

Deleting a service from a working SFC

In this use case we will start with a SFC with multiple services running and functional. Once everything is steady state, we will delete a service from SFC by updating the service function chain and make sure traffic is going through only the services in action.

Test Steps	**Test Results**
Create LEFT-1 network	LEFT-1 network created successfully
Create LEFT-1 subnet with following attributes --gateway 1.1.0.1 --subnet-range 1.1.0.0/24 --host-route	LEFT-1 subnet created successfully

destination=192.168.10.0/24,gateway=1.1.0.1	
Add LEFT-1 subnet to the openstack router	Subnet added successfully
Create RIGHT-1 network	RIGHT-1 network created successfully
Create RIGHT-1 subnet with following attributes --gateway 2.2.0.1 --subnet-range 2.2.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.2.0.1 --host-route destination=128.0.0.0/1,gateway=2.2.0.1	RIGHT-1 subnet created successfully
Add RIGHT-1 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-1 network	Port created successfully
Create a port on RIGHT-1 network	Port created successfully
Create a Service VM with ports from LEFT-1 and RIGHT-1 networks with OpenWRTSI1 image	Service VM created successfully
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 192.168.10.0/24 --l7-parameters logical_source_network=SRC-NET-ID, logical_destination_network=DEST-NET-ID	Flow Classifier created successfully
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully
Create LEFT-2 network	LEFT-2 network created successfully
Create LEFT-2 subnet with following attributes --gateway 1.2.0.1 --subnet-range 1.2.0.0/24 --host-route destination=192.168.10.0/24,gateway=1.1.0.1	LEFT-2 subnet created successfully
Add LEFT-2 subnet to the openstack router	Subnet added successfully
Create RIGHT-2 network	RIGHT-2 network created successfully
Create RIGHT-2 subnet with following attributes --gateway 2.2.0.1 --subnet-range 2.2.0.0/24	RIGHT-2 subnet created successfully

--host-route destination=0.0.0.0/1,gateway=2.2.0.1 --host-route destination=128.0.0.0/1,gateway=2.2.0.1	
Add RIGHT-2 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-2 network	Port created successfully
Create a port on RIGHT-2 network	Port created successfully
Create a Service VM with ports from LEFT-2 and RIGHT-2 networks with OpenWRTSI2 image	Service VM created successfully
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully
Ping the Destination VM from the Source VM	All Ping successful as they are going through the router
Create a port chain with the flow classifier and both the port pair group	Service Function Chain created successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	All Ping successful except 8.8.8.1, 8.8.8.2. Blocked by Service VMs
Update the port chain by removing the second port pair group	Service Function Chain updated successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	All Ping successful except 8.8.8.1 Blocked by the first Service VM. 8.8.8.2 is not blocked any more and goes through the router as we removed this service from the service function chain.

Scale Test Cases

Scale the number of SFCs in parallel

In this use case we would write a script with the base topology and loop the SFC creation for X number of SFCs of the same type. The number can go from 2 to 5 to 10 to 100 depending on a small configuration change. This will be primarily used to load balance the traffic coming in from the customer site and scale a service to a very high level.

Test Steps	**Test Results**
Create an openstack router	Openstack router created successfully.
Create source network with following attributes	Source network created successfully
Create source subnet with following attributes --gateway 192.168.10.1 --subnet-range 192.168.10.0/24	Source subnet created successfully
Add the source subnet to the openstack router	Subnet added successfully
Create a port on the access network	Port created successfully
Create a port on the source network	Port created successfully
Create Source VM with ports from access network and source network	Source VM created successfully
Ping the gateway address from Source VM	Ping successful
Create destination network with following attributes	Destination network created successfully
Create destination subnet with following attributes --gateway 192.168.20.1 --subnet-range 192.168.20.0/24	Destination subnet created successfully
Add the destination subnet to the openstack router	Subnet added successfully
Create a port on the access network	Port created successfully
Create a port on the destination network	Port created successfully
Create a VM with ports from access network and destination network	Destination VM created successfully

Ping the gateway address from Destination VM	Ping successful
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	Ping successful
Create LEFT-1 network	LEFT-1 network created successfully
Create LEFT-1 subnet with following attributes --gateway 1.1.0.1 --subnet-range 1.1.0.0/24 --host-route destination=192.168.10.0/24,gateway=1.1.0.1	LEFT-1 subnet created successfully
Add LEFT-1 subnet to the openstack router	Subnet added successfully
Create RIGHT-1 network	RIGHT-1 network created successfully
Create RIGHT-1 subnet with following attributes --gateway 2.1.0.1 --subnet-range 2.1.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.1.0.1 --host-route destination=128.0.0.0/1,gateway=2.1.0.1	RIGHT-1 subnet created successfully
Add RIGHT-1 subnet to the openstack router	Subnet added successfully
Define X the scale number for no. of parallel service VMs we want to scale up to in the service function chain. They can be 2, 5, 10 or even 100.	Scale number defined properly against variable X.
Create X no. of ports on LEFT-1 network.	Ports created successfully
Create X no. of ports on RIGHT-1 network	Ports created successfully
Create X no. of Service VMs with ports from LEFT-1 and RIGHT-1 networks with OpenWRTSI1 image	Service VMs created successfully
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 192.168.10.0/24 --l7-parameters logical_source_network=SRC-NET-ID, logical_destination_network=DEST-NET-ID	Flow Classifier created successfully
Create X no. of port pairs with ports used for creating the X no. of service VMs	Port Pairs create successfully
Create a port pair group with the port pairs created	Port Pair Group create successfully

Ping multiple IP address on Destination VM from the Source VM (8.8.8.x) range	Ping successful for all IP address
Create a port chain with the flow classifier and port pair group	Service Function Chain created successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	Ping successful and going through different newly created SFCs based on hash except for 8.8.8.1 which is blocked

Scale the number of SFCs in serial

In this use case we would write a script with the base topology and loop the SFC creation for X number of SFCs of different types. For every SFC chain we will be creating new LEFT and RIGHT networks. The number can go from 2 to 5 to 10 to 100 depending on a small configuration change. This will be primarily used to introduce multiple services to the same traffic coming in from the customer site.

Test Steps	**Test Results**
Create an openstack router	Openstack router created successfully.
Create source network with following attributes	Source network created successfully
Create source subnet with following attributes --gateway 192.168.10.1 --subnet-range 192.168.10.0/24	Source subnet created successfully
Add the source subnet to the openstack router	Subnet added successfully
Create a port on the access network	Port created successfully
Create a port on the source network	Port created successfully
Create Source VM with ports from access network and source network	Source VM created successfully
Ping the gateway address from Source VM	Ping successful
Create destination network with following attributes	Destination network created successfully
Create destination subnet with following attributes --gateway 192.168.20.1 --subnet-range 192.168.20.0/24	Destination subnet created successfully
Add the destination subnet to the openstack router	Subnet added successfully
Create a port on the access network	Port created successfully
Create a port on the destination network	Port created successfully
Create a VM with ports from access network and destination network	Destination VM created successfully
Ping the gateway address from Destination VM	Ping successful

Ping the Destination VM from the Source VM	Ping successful
Define X the scale number to accomplish no. of services in a serial chain. X can be of any value, like 2, 5, 10 or even 100.	Scale number defined properly against variable X.
Create LEFT-X networks	LEFT-X networks created successfully
Create LEFT-X subnet with following attributes --gateway 1.X.0.1 --subnet-range 1.X.0.0/24 --host-route destination=192.168.10.0/24,gateway=1.X.0.1	LEFT-X subnets created successfully
Add LEFT-X subnets to the openstack router	Subnets added successfully
Create RIGHT-X networks	RIGHT-X networks created successfully
Create RIGHT-X subnet with following attributes --gateway 2.X.0.1 --subnet-range 2.X.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.X.0.1 --host-route destination=128.0.0.0/1,gateway=2.X.0.1	RIGHT-1 subnet created successfully
Add RIGHT-X subnets to the openstack router	Subnets added successfully
Create a port on LEFT-X network	Ports created successfully
Create a port on RIGHT-X network	Ports created successfully
Create Service VMs with ports from LEFT-X and RIGHT-X networks with OpenWRTSIX image	Service VMs created successfully
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 192.168.10.0/24 --l7-parameters logical_source_network=SRC-NET-ID, logical_destination_network=DEST-NET-ID	Flow Classifier created successfully
Create port pairs with ports used for creating the service VMs	Port Pair create successfully
Create a port pair group with the port pairs created	Port Pair Group create successfully
Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	All Ping successful except 8.8.8.1 to 8.8.8.X Blocked by X Service VMs

Scale the number of SFCs in serial as well as parallel

In this use case we would write a script with the base topology and loop the SFC creation for X number of SFCs of different types and Y number of SFCs of same type. For every SFC chain we will be creating new LEFT and RIGHT networks. The value of number X and Y can go from 2 to 5 to 10 to 100 depending on a small configuration change. This scale test case can cover anything required for SFC scaling based on customer needs.

Test Steps	**Test Results**
Create an openstack router	Openstack router created successfully.
Create source network with following attributes	Source network created successfully
Create source subnet with following attributes --gateway 192.168.10.1 --subnet-range 192.168.10.0/24	Source subnet created successfully
Add the source subnet to the openstack router	Subnet added successfully
Create a port on the access network	Port created successfully
Create a port on the source network	Port created successfully
Create Source VM with ports from access network and source network	Source VM created successfully
Ping the gateway address from Source VM	Ping successful
Create destination network with following attributes	Destination network created successfully
Create destination subnet with following attributes --gateway 192.168.20.1 --subnet-range 192.168.20.0/24	Destination subnet created successfully
Add the destination subnet to the openstack router	Subnet added successfully
Create a port on the access network	Port created successfully
Create a port on the destination network	Port created successfully
Create a VM with ports from access network and destination network	Destination VM created successfully
Ping the gateway address from Destination VM	Ping successful

Ping the Destination VM from the Source VM	Ping successful
Define X the scale number to accomplish no. of services in the chain. X can be of any value, like 2, 5, 10 or even 100.	Scale number defined properly against variable X.
Define Y the scale number for no. of parallel service VMs in the chain. Y can be of any value, like 2, 5, 10 or even 100.	Scale number defined properly against variable Y.
Create LEFT-X networks	LEFT-X networks created successfully
Create LEFT-X subnet with following attributes --gateway 1.X.0.1 --subnet-range 1.X.0.0/24 --host-route destination=192.168.10.0/24,gateway=1.X.0.1	LEFT-X subnets created successfully
Add LEFT-X subnets to the openstack router	Subnets added successfully
Create RIGHT-X networks	RIGHT-X networks created successfully
Create RIGHT-X subnet with following attributes --gateway 2.X.0.1 --subnet-range 2.X.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.X.0.1 --host-route destination=128.0.0.0/1,gateway=2.X.0.1	RIGHT-1 subnet created successfully
Add RIGHT-X subnets to the openstack router	Subnets added successfully
Create Y no. of ports on every LEFT-X network	Ports created successfully
Create Y no. of ports on every RIGHT-X network	Ports created successfully
Create Service VMs with ports from LEFT-X-Y and RIGHT-X-Y networks with OpenWRTSIX image	Service VMs created successfully
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 192.168.10.0/24 --l7-parameters logical_source_network=SRC-NET-ID, logical_destination_network=DEST-NET-ID	Flow Classifier created successfully
Create port pairs with ports used for creating the service VMs	Port Pair create successfully
Create a port pair group with the port pairs created	Port Pair Group create successfully

Ping multiple IP address on Destination VM from the Source VM (8.8.8.1-8) range	All Ping successful except 8.8.8.1 to 8.8.8.X Blocked by X Service VMs. Also the traffic should be load shared across all the service VMs which are deployed in parallel for a certain service.

Neutron Trunking

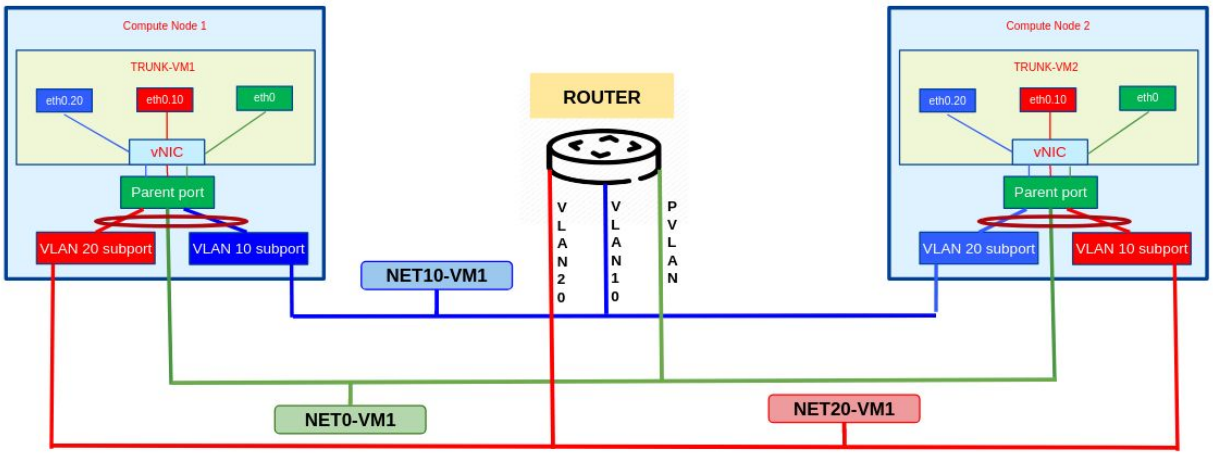
The network trunk service allows multiple networks to be connected to an instance using a single virtual NIC (vNIC). Multiple networks can be presented to an instance by connecting it to a single port. Users can create a port, associate it with a trunk, and launch an instance on that port. Users can dynamically attach and detach additional networks without disrupting the operation of the instance.

Every trunk has a parent port and can have any number of subports. The parent port is the port that the trunk is associated with. Users create instances and specify the parent port of the trunk when launching instances attached to a trunk.

Setup Details

The setup for testing neutron trunking capabilities in openstack is created with three networks in place all connected to a router. Of these one Vlan acts like a native vlan and the other two as tagged vlans. The VMs spun in an openstack environment will have trunk ports, where the native vlan will be associated to default namespace on the VM. For the Vlans which are tagged, we create separate namespaces to have their own routing tables. Also we have VMs created in each of these networks on regular non trunk ports. Tests are done in this topology for intra and inter vlan reachability.

Setup Diagram



Functional Test Cases

Base Topology and Configurations

TRUNK-VM1 Configuration

This configuration is applied on the Trunk-VM1 to create required namespaces to isolate different VLANs of the Trunk port of the compute node

```
ip netns add cats
ip link add link eth1 name eth1.10 type vlan id 10
ip link set eth1.10 netns cats
ip netns exec cats ifconfig eth1.10 hw ether fa:16:3e:bc:d5:38
ip netns exec cats /sbin/dhclient -1 eth1.10

ip netns add dogs
ip link add link eth1 name eth1.20 type vlan id 20
ip link set eth1.20 netns dogs
ip netns exec dogs ifconfig eth1.20 hw ether fa:16:3e:bc:d5:39
ip netns exec dogs /sbin/dhclient -1 eth1.20
```

TRUNK-VM2 Configuration

This configuration is applied on the Trunk-VM1 to create required namespaces to isolate different VLANs of the Trunk port of the compute node

```
ip netns add cats
ip link add link eth1 name eth1.10 type vlan id 10
ip link set eth1.10 netns cats
ip netns exec cats ifconfig eth1.10 hw ether fa:16:3e:1b:a1:a1
ip netns exec cats /sbin/dhclient -1 eth1.10

ip netns add dogs
ip link add link eth1 name eth1.20 type vlan id 20
ip link set eth1.20 netns dogs
ip netns exec dogs ifconfig eth1.20 hw ether fa:16:3e:1b:a1:a2
ip netns exec dogs /sbin/dhclient -1 eth1.20
```

Steps to build base Topology

Test Steps	**Test Results**
Create an openstack router	Openstack router created successfully.
Create NET0 network	NET0 network created successfully
Create NET0 subnet with following attributes --gateway 192.168.0.1 --subnet-range 192.168.0.0/24	NET0 subnet created successfully
Add the NET0 subnet to the openstack router	Subnet added successfully
Create two ports on the access network	Ports created successfully
Create two ports on the NET0 network	Ports created successfully
Create two trunks using ports on NET0	Trunks created successfully
Create two VMs with ports from access network and NET0 network	VMs TRUNK-VM1 and TRUNK-VM2 created successfully
Ping the gateway address from both the VMs	Pings successful
Create one more port on NET0	Port created successfully
Create a VM using this port	NET0-VM1 VM created successfully
Create NET10 network	NET10 network created successfully
Create NET10 subnet with following attributes --gateway 192.168.10.1 --subnet-range 192.168.10.0/24	NET10 subnet created successfully
Add the NET10 subnet to the openstack router	Subnet added successfully
Create two ports on NET10 with specific mac-address --mac-address fa:16:3e:bc:d5:38 --mac-address fa:16:3e:1b:a1:a1	Ports created successfully
Add one port each to the two trunks created before segmentation-type=vlan,segmentation-id=10	Ports addition to trunks successful
Create one more port on NET10	Port created successfully

Create a VM using this port	NET20-VM1 VM created successfully
Create NET20 network	NET20 network created successfully
Create NET20 subnet with following attributes --gateway 192.168.20.1 --subnet-range 192.168.20.0/24	NET20 subnet created successfully
Add the NET20 subnet to the openstack router	Subnet added successfully
Create two ports on NET20 with specific mac-address --mac-address fa:16:3e:bc:d5:39 --mac-address fa:16:3e:1b:a1:a2	Ports created successfully
Add one port each to the two trunks created before segmentation-type=vlan,segmentation-id=20	Ports addition to trunks successful
Create one more port on NET20	Port created successfully
Create a VM using this port	NET20-VM1 VM created successfully
Apply config to TRUNK-VM!	Configuration applied
Apply config to TRUNK-VM2	Configuration applied
From each namespace default, cats and dogs ping the default gateways	All pings should be successful

Nodes reachability in native Vlan

In this use case, we test the connectivity between the parent port on both the trunk VMs and make sure they are also reachable from the non trunk VM spun on the same network.

Test Steps	**Test Results**
Ping the IP address of NET0-VM1 from default namespace of TRUNK-VM1	Ping should be successful
Ping the IP address of NET0-VM1 from default namespace of TRUNK-VM2	Ping should be successful

Intra Vlan nodes reachability on tagged Vlans

In this use case we test the connectivity between the nodes with in Vlan 10 and Vlan 20.

Test Steps	**Test Results**
Ping the IP address of NET10-VM1 from cats namespace of TRUNK-VM1	Ping should be successful
Ping the IP address of NET20-VM1 from dogs namespace of TRUNK-VM1	Ping should be successful
Ping the IP address of NET10-VM1 from cats namespace of TRUNK-VM2	Ping should be successful
Ping the IP address of NET20-VM1 from dogs namespace of TRUNK-VM2	Ping should be successful
Ping the cats namespace IP address of TRUNK-VM2 from cats namespace of TRUNK-VM1	Ping should be successful
Ping the dogs namespace IP address of TRUNK-VM2 from dogs namespace of TRUNK-VM1	Ping should be successful

Inter Vlan nodes reachability between all Vlans

In this use case we test the reachability of all nodes from all nodes on all vlan networks

Test Steps	**Test Results**
Ping the IP address of NET10-VM1 from default namespace of TRUNK-VM1	Ping should be successful

Ping the IP address of NET20-VM1 from default namespace of TRUNK-VM1	Ping should be successful
Ping the IP address of NET10-VM1 from default namespace of TRUNK-VM2	Ping should be successful
Ping the IP address of NET20-VM1 from default namespace of TRUNK-VM2	Ping should be successful
Ping the IP address of NET0-VM1 from cats namespace of TRUNK-VM1	Ping should be successful
Ping the IP address of NET0-VM1 from dogs namespace of TRUNK-VM1	Ping should be successful
Ping the IP address of NET0-VM1 from cats namespace of TRUNK-VM2	Ping should be successful
Ping the IP address of NET0-VM1 from dogs namespace of TRUNK-VM2	Ping should be successful
Ping the cats namespace IP address of TRUNK-VM2 from dogs namespace of TRUNK-VM1	Ping should be successful
Ping the dogs namespace IP address of TRUNK-VM2 from cats namespace of TRUNK-VM1	Ping should be successful

Adding a Vlan interface to trunk port

In this use case we make sure that a new Vlan port can be created and added to an existing trunk port of an existing VM and test reachability of it from all other places.

Test Steps	**Test Results**
Create NET30 network	NET30 network created successfully
Create NET30 subnet with following attributes --gateway 192.168.30.1 --subnet-range 192.168.30.0/24	NET30 subnet created successfully
Add the NET30 subnet to the openstack router	Subnet added successfully
Create two ports on NET30 with specific mac-address --mac-address fa:16:3e:bc:d5:3a --mac-address fa:16:3e:1b:a1:a3	Ports created successfully
Add one port each to the two trunks created before	Ports addition to trunks successful

segmentation-type=vlan,segmentation-id=30	
Configure TRUNK-VM1 with new namespace with below commands ip netns add frogs ip link add link eth1 name eth1.30 type vlan id 30 ip link set eth1.30 netns frogs ip netns exec frogs ifconfig eth1.30 hw ether fa:16:3e:bc:d5:3a ip netns exec frogs /sbin/dhclient -1 eth1.30	The new interface should get an IP address
Configure TRUNK-VM2 with new namespace with below commands ip netns add frogs ip link add link eth1 name eth1.30 type vlan id 30 ip link set eth1.30 netns frogs ip netns exec frogs ifconfig eth1.30 hw ether fa:16:3e:1b:a1:a3 ip netns exec frogs /sbin/dhclient -1 eth1.30	The new interface should get an IP address
Ping IP address of default gateway from frogs namespace on TRUNK-VM1	Ping should be successful
Ping IP address of default gateway from frogs namespace on TRUNK-VM2	Ping should be successful
Ping the IP address of frogs namespace of TRUNK-VM1 from NET0-VM1	Ping should be successful
Ping the IP address of frogs namespace of TRUNK-VM1 from NET10-VM1	Ping should be successful
Ping the IP address of frogs namespace of TRUNK-VM1 from NET20-VM1	Ping should be successful
Ping the IP address of frogs namespace of TRUNK-VM2 from NET0-VM1	Ping should be successful
Ping the IP address of frogs namespace of TRUNK-VM2 from NET10-VM1	Ping should be successful
Ping the IP address of frogs namespace of TRUNK-VM2 from NET20-VM1	Ping should be successful
Ping the IP address of frogs namespace of TRUNK-VM1 from TRUNK-VM1 default namespace	Ping should be successful

Ping the IP address of frogs namespace of TRUNK-VM1 from TRUNK-VM1 default namespace	Ping should be successful
Ping the IP address of frogs namespace of TRUNK-VM1 from TRUNK-VM1 default namespace	Ping should be successful
Ping the IP address of frogs namespace of TRUNK-VM2 from TRUNK-VM1 default namespace	Ping should be successful
Ping the IP address of frogs namespace of TRUNK-VM2 from TRUNK-VM1 default namespace	Ping should be successful
Ping the IP address of frogs namespace of TRUNK-VM2 from TRUNK-VM1 default namespace	Ping should be successful

Deleting a Vlan interface from trunk port

In this use case we delete a Vlan port from a trunk interface and make sure the IP address is not reachable any more from outside work.

Test Steps	**Test Results**
Delete one port each from the two trunks created before segmentation-type=vlan,segmentation-id=30	Ports deletion from trunks successful
Ping the IP address of frogs namespace of TRUNK-VM1 from NET0-VM1	Ping should Fail
Ping the IP address of frogs namespace of TRUNK-VM1 from NET10-VM1	Ping should Fail
Ping the IP address of frogs namespace of TRUNK-VM1 from NET20-VM1	Ping should Fail
Ping the IP address of frogs namespace of TRUNK-VM2 from NET0-VM1	Ping should Fail
Ping the IP address of frogs namespace of TRUNK-VM2 from NET10-VM1	Ping should Fail
Ping the IP address of frogs namespace of TRUNK-VM2 from NET20-VM1	Ping should Fail
Ping the IP address of frogs namespace of TRUNK-VM1 from TRUNK-VM1 default namespace	Ping should Fail
Ping the IP address of frogs namespace of	Ping should Fail

TRUNK-VM1 from TRUNK-VM1 default namespace	
Ping the IP address of frogs namespace of TRUNK-VM1 from TRUNK-VM1 default namespace	Ping should Fail
Ping the IP address of frogs namespace of TRUNK-VM2 from TRUNK-VM1 default namespace	Ping should Fail
Ping the IP address of frogs namespace of TRUNK-VM2 from TRUNK-VM1 default namespace	Ping should Fail
Ping the IP address of frogs namespace of TRUNK-VM2 from TRUNK-VM1 default namespace	Ping should Fail

Scale Test Cases

VLAN Scale on Trunk port Use Case

In this use case we will use the base configuration of Vlan Trunk functional test and scale on the no. of sub-ports we will add to the trunk port. We can define a variable X for no. of vlans or sub-ports and sequentially scale. The variable X can pick any value user configures like 2, 5, 10 or 100 and can build the configuration accordingly. The validation also needs to be done from every vlan to every other vlan.

TRUNK-VM1 Configuration

```
NOOFVLANS=105
for (( VID=101; VID<=$NOOFVLANS; VID++ ))
do
   HEXVID=$(printf "%x\n" $VID)
   ip netns add "cust-$VID"
   ip link add link eth1 name eth1.$VID type vlan id $VID
   ip link set eth1.$VID netns "cust-$VID"
   ip netns exec "cust-$VID" ifconfig eth1.$VID hw ether fa:16:3e:bc:d5:$HEXVID
   ip netns exec "cust-$VID" /sbin/dhclient -1 eth1.$VID
done
```

TRUNK-VM2 Configuration

```
NOOFVLANS=105
for (( VID=101; VID<=$NOOFVLANS; VID++ ))
do
   HEXVID=$(printf "%x\n" $VID)
   ip netns add "cust-$VID"
   ip link add link eth1 name eth1.$VID type vlan id $VID
   ip link set eth1.$VID netns "cust-$VID"
   ip netns exec "cust-$VID" ifconfig eth1.$VID hw ether fa:16:3e:1b:a1:$HEXVID
   ip netns exec "cust-$VID" /sbin/dhclient -1 eth1.$VID
done
```

Steps to build Scale Topology

Test Steps	**Test Results**
Create an openstack router	Openstack router created successfully.
Create NET0 network	NET0 network created successfully
Create NET0 subnet with following attributes --gateway 192.168.0.1 --subnet-range 192.168.0.0/24	NET0 subnet created successfully
Add the NET0 subnet to the openstack router	Subnet added successfully
Create two ports on the access network	Ports created successfully
Create two ports on the NET0 network	Ports created successfully
Create two trunks using ports on NET0	Trunks created successfully
Create two VMs with ports from access network and NET0 network	VMs TRUNK-VM1 and TRUNK-VM2 created successfully
Ping the gateway address from both the VMs	Pings successful
Define X the scale number to accomplish no. of vlans supported on a trunk port connected to a compute noden. X can be of any value, like 2, 5, 10 or even 100.	Scale number defined properly against variable X.
Create NET-X network	NET-X network created successfully
Create NET-X subnet with following attributes --gateway 192.168.X.1 --subnet-range 192.168.X.0/24	NET-X subnet created successfully
Add the NET-X subnet to the openstack router	Subnet added successfully
Create two ports on NET-X with specific mac-address --mac-address fa:16:3e:bc:d5:X --mac-address fa:16:3e:1b:a1:X	Ports created successfully
Add one port each to the two trunks created before segmentation-type=vlan,segmentation-id=X	Ports addition to trunks successful
Create namespace configuration for	

TRUNK-VM1 using script	
Apply config to TRUNK-VM	Configuration applied
Apply config to TRUNK-VM2	Configuration applied

Steps for Validation

Test Steps	**Test Results**
Ping IP address of all namespaces of TRUNK-VM1 from default namespace of TRUNK-VM2	Ping should be successful
Ping IP address of all namespaces of TRUNK-VM2 from default namespace of TRUNK-VM1	Ping should be successful
Ping IP address of all namespaces of TRUNK-VM1 from default namespace of TRUNK-VM1	Ping should be successful
Ping IP address of all namespaces of TRUNK-VM2 from default namespace of TRUNK-VM2	Ping should be successful
Ping IP address of all namespaces of TRUNK-VM1 from corresponding namespaces of TRUNK-VM2	Ping should be successful
Ping IP address of all namespaces of TRUNK-VM2 from corresponding namespaces of TRUNK-VM1	Ping should be successful

SVI

Traditionally, switches send traffic only to hosts within the same broadcast domain (Single VLAN) and routers handled traffic between different broadcast domains (Different VLANs). This meant that network devices in different broadcast domains could not communicate without a router. With SVIs the switch will use virtual Layer 3 interface to route traffic to other Layer 3 interface

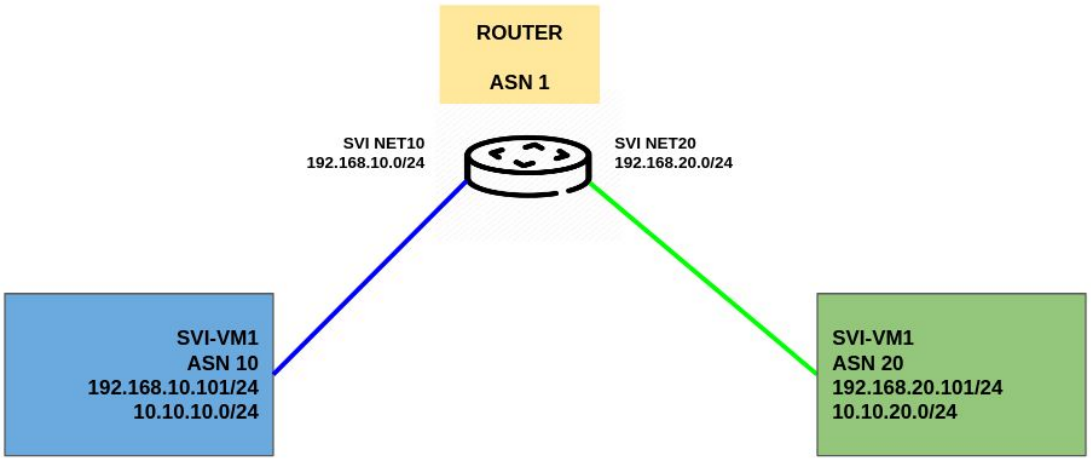
Switched virtual interface – SVI. An SVI is normally found on switches (Layer 3 and Layer 2). With SVIs the switch recognizes the packet destinations that are local to the sending VLAN and switches those packets and packets destined for different VLANs are routed.

An SVI cannot be activated unless the VLAN itself is created and at least one physical port is associated and active in that VLAN. In openstack case SVI is created on the leaf node only if a vm exists on a connected compute node.

Setup Details

The setup used for SVI testing constitutes of two networks connected to a router. We have BGP turned on the ACI side for corresponding network SVIs. We also have two VMs created and connected to each network. We have bird BGP routing stack running on the VMs to exchange BGP routes. We use secondary addresses to advertise prefixes on BGP connections. As both SVIs belong to different ASNs, the routes get exchanged between the hosts. We add static routes for reachability.

Setup Diagram



Functional Test Cases

Base Topology and Configurations

SVI-VM1 Configuration

This configuration is applied on SVI-VM1 to create dummy networks. The dummy networks are advertised through BGP to Cisco ACI. Static routes are also added to reach the routes advertised by BGP running on SVI-VM2.

```
ip addr add 10.10.10.1/24 dev eth0
ip addr add 10.10.10.2/24 dev eth0
ip addr add 10.10.10.3/24 dev eth0
ip addr add 10.10.10.4/24 dev eth0
ip addr add 10.10.10.5/24 dev eth0

ip route add 192.168.20.0/24 via 192.168.10.1
ip route add 10.10.20.0/24 via 192.168.10.1
```

SVI-VM2 Configuration

This configuration is applied on SVI-VM2 to create dummy networks. The dummy networks are advertised through BGP to Cisco ACI. Static routes are also added to reach the routes advertised by BGP running on SVI-VM1.

```
ip addr add 10.10.20.1/24 dev eth0
ip addr add 10.10.20.2/24 dev eth0
ip addr add 10.10.20.3/24 dev eth0
ip addr add 10.10.20.4/24 dev eth0
ip addr add 10.10.20.5/24 dev eth0

ip route add 192.168.10.0/24 via 192.168.20.1
ip route add 10.10.10.0/24 via 192.168.20.1
```

SVI-VM1 Bird Configuration

Bird configuration to be used on SVI-VM1

```
router id 199.199.199.201;
#protocol kernel {
#       persist;              # Don't remove routes on bird shutdown
#       scan time 20;         # Scan kernel routing table every 20 seconds
#       export all;           # Default is export none
#}
# This pseudo-protocol watches all interface up/down events.
protocol device {
       scan time 10;         # Scan interfaces every 10 seconds
}
protocol static {
          ipv4;
       route 10.10.10.1/32 via 192.168.10.101;
       route 10.10.10.2/32 via 192.168.10.101;
       route 10.10.10.3/32 via 192.168.10.101;
       route 10.10.10.4/32 via 192.168.10.101;
       route 10.10.10.5/32 via 192.168.10.101;
}
protocol bgp {
       description "My BGP uplink";
       local as 10;
       neighbor 192.168.10.199 as 1;
       ipv4 {
             import all;
             export all;
       };
       source address 192.168.10.101;    # What local address we use for the TCP connection
}
protocol bgp {
       description "My BGP uplink";
       local as 10;
       neighbor 192.168.10.200 as 1;
       ipv4 {
          import all;
          export all;
       };
       source address 192.168.10.101;    # What local address we use for the TCP connection
}
```

SVI-VM2 Bird Configuration

Bird configuration to be used on SVI-VM2

```
router id 199.199.199.202;
#protocol kernel {
#       persist;              # Don't remove routes on bird shutdown
#       scan time 20;         # Scan kernel routing table every 20 seconds
#       export all;           # Default is export none
#}
# This pseudo-protocol watches all interface up/down events.
protocol device {
      scan time 10;         # Scan interfaces every 10 seconds
}
protocol static {
         ipv4;
      route 10.10.20.1/32 via 192.168.20.101;
      route 10.10.20.2/32 via 192.168.20.101;
      route 10.10.20.3/32 via 192.168.20.101;
      route 10.10.20.4/32 via 192.168.20.101;
      route 10.10.20.5/32 via 192.168.20.101;
}
protocol bgp {
      description "My BGP uplink";
      local as 20;
      neighbor 192.168.20.199 as 1;
      ipv4 {
            import all;
            export all;
      };
      source address 192.168.20.101;    # What local address we use for the TCP connection
}
protocol bgp {
      description "My BGP uplink";
      local as 20;
      neighbor 192.168.20.200 as 1;
      ipv4 {
            import all;
            export all;
      };
      source address 192.168.20.101;    # What local address we use for the TCP connection
}
```

Steps to build base Topology

Test Steps	**Test Results**
Create an openstack router	Openstack router created successfully.
Create NET10 network with following attributes --provider:network_type vlan --apic:svi True --apic:bgp_enable True --apic:bgp_asn 10	The network should be successfully created without any errors
Create subnet for access network with following attributes --gateway 192.168.10.1 --subnet-range 192.168.10.0/24	The subnet should be created successfully without any errors
Create SVI ports on all leaf switches with the following attributes. Change the IP address for each node. --device-owner apic:svi --fixed-ip subnet=”subnet-id”,ip-address=192.168.10.199	The SVI ports for all the leaf switches should be created successfully without any errors
Add the NET10 subnet to the openstack router	Subnet added successfully
Create a port on the access network	Port created successfully
Create a port on the NET10 network	Port created successfully
Create VM with ports from access network and NET10 network	SVI-VM1 VM created successfully
Ping the gateway address from SVI-VM1	Ping successful
Create NET20 network with following attributes --provider:network_type vlan --apic:svi True --apic:bgp_enable True --apic:bgp_asn 20	The network should be successfully created without any errors
Create subnet for access network with following attributes --gateway 192.168.20.1 --subnet-range 192.168.20.0/24	The subnet should be created successfully without any errors

Create SVI ports on all leaf switches with the following attributes. Change the IP address for each node. --device-owner apic:svi --fixed-ip subnet=”subnet-id”,ip-address=192.168.20.199	The SVI ports for all the leaf switches should be created successfully without any errors
Add the NET20 subnet to the openstack router	Subnet added successfully
Create a port on the access network	Port created successfully
Create a port on the NET20 network	Port created successfully
Create VM with ports from access network and NET20 network	SVI-VM1 VM created successfully
Ping the gateway address from SVI-VM2	Ping successful
Apply config to SVI-VM1	Configuration applied
Apply config to SVI-VM2	Configuration applied

Reachability between SVI VMs

In this test case, we make sure the VMs are spun properly and they are reachable across the router created as part of the topology.

Test Steps	**Test Results**
Ping SVI-VM1 IP address from SVI-VM2	Ping should be successful
Ping SVI-VM2 IP address from SVI-VM1	Ping should be successful

BGP connectivity between SVI VMs and ACI

In this use case we configure bird and run bird BGP routing stack on each of the VMs. We validate that the BGP peering between the VMs and the ACI is in place. We also validate that the routes advertised from one VM have reached the other VM and vice versa.

Test Steps	**Test Results**
Copy bird configuration file of SVI-VM1 to SVI-VM1	Copy successful
Copy bird configuration file of SVI-VM2 to SVI-VM2	Copy successful
Start bird application on SVI-VM1 with below command	Bird application started successfully

bird -c bird-eth1.conf	
Start bird application on SVI-VM2 with below command bird -c bird-eth1.conf	Bird application started successfully
Check established neighbors using below command on SVI-VM1 birdc show protocol	Neighbor should be established at least to one of the leaf nodes
Check established neighbors using below command on SVI-VM2 birdc show protocol	Neighbor should be established at least to one of the leaf nodes
Check BGP routes advertised as well as received on both SVI-VM1 using below command birdc show route	The prefixes advertised as well as received should be shown in this output
Check BGP routes advertised as well as received on both SVI-VM2 using below command birdc show route	The prefixes advertised as well as received should be shown in this output

Reachability of BGP advertised routes across ACI SVI interfaces

We use secondary addresses on each of the VMs for creating route prefixes to be advertised to BGP peers. We test reachability of these prefixes from both the VMs in this use case across the ACI fabric.

Test Steps	**Test Results**
Ping 10.10.20.1-5 from SVI-VM1	Ping should be successful
Ping 10.10.10.1-5 from SVI-VM2	Ping should be successful

Scale Test Cases

SVI Scalability Use Case

This scale use case has to test the number of SVIs we can configure and test on Cisco ACI. We will use a variable X here whose value can be anything from 2, 5, 10 or 100 and the script can configure so many SVIs and also spin VM on each SVI and run BGP with unique ASN. The advertised routes should also be a variation of X, so that one parameter can cover all of the aspects of the SVI testing. The connectivity on advertised BGP prefixes should also be validated as part of this testing.

SVI-VM-X Configuration

We need to apply this configuration on the VM we create on each SVI network. Here is the SVID changes for each VM based on the SVI network the VM belongs to.

ip addr add 11.10.SVID.1/24 dev eth0
ip addr add 11.10.SVID.2/24 dev eth0
ip addr add 11.10.SVID.3/24 dev eth0
ip addr add 11.10.SVID.4/24 dev eth0
ip addr add 11.10.SVID.5/24 dev eth0

ip route add 192.168.0.0/16 via 192.168.SVID.1
ip route add 11.10.0.0/16 via 192.168.SVID.1

SVI-VM-X Bird Configuration

Even here we need to change the value of SVID based on the VM which belongs to the SVI network it is in.

```
router id 199.199.SVID.201;
#protocol kernel {
#       persist;              # Don't remove routes on bird shutdown
#       scan time 20;         # Scan kernel routing table every 20 seconds
#       export all;           # Default is export none
#}
# This pseudo-protocol watches all interface up/down events.
protocol device {
      scan time 10;         # Scan interfaces every 10 seconds
}
protocol static {
      ipv4;
      route 11.10.SVID.1/32 via 192.168.SVID.101;
      route 11.10.SVID.2/32 via 192.168.SVID.101;
      route 11.10.SVID.3/32 via 192.168.SVID.101;
      route 11.10.SVID.4/32 via 192.168.SVID.101;
      route 11.10.SVID.5/32 via 192.168.SVID.101;
}
protocol bgp {
      description "My BGP uplink";
      local as SVID;
      neighbor 192.168.SVID.199 as 1;
      ipv4 {
            import all;
            export all;
      };
      source address 192.168.SVID.101;    # What local address we use for the TCP connection
}
protocol bgp {
      description "My BGP uplink";
      local as SVID;
      neighbor 192.168.SVID.200 as 1;
      ipv4 {
            import all;
            export all;
      };
      source address 192.168.SVID.101;    # What local address we use for the TCP connection
}
```

Steps to build Scale Topology

Test Steps	**Test Results**
Create an openstack router	Openstack router created successfully.
Define X the scale number to accomplish no. of SVIs we want to test with. X can be of any value, like 2, 5, 10 or even 100. We create X no. of SVI networks. Create a VM in every network and run BGP on every VM to connect to the ACI network on unique ASN	Scale number defined properly against variable X.
Create NET-X network with following attributes --provider:network_type vlan --apic:svi True --apic:bgp_enable True --apic:bgp_asn X	The network should be successfully created without any errors
Create subnet for access network with following attributes --gateway 192.168.X.1 --subnet-range 192.168.X.0/24	The subnet should be created successfully without any errors
Create SVI ports on all leaf switches with the following attributes. Change the IP address for each node. --device-owner apic:svi --fixed-ip subnet=”subnet-id”,ip-address=192.168.X.199	The SVI ports for all the leaf switches should be created successfully without any errors
Add the NET-X subnet to the openstack router	Subnet added successfully
Create a port on the access network	Port created successfully
Create a port on the NET-X network	Port created successfully
Create VM with ports from access network and NET-X network	SVI-NET-X-VM1 VM created successfully
Apply config to SVI-NET-X-VM1	Configuration applied
Copy bird configuration file of SVI-NET-X-VM1 to SVI-NET-X-VM1	Copy successful
Start bird application on SVI-NET-X-VM1 with below command	Bird application started successfully

bird -c bird-eth1.conf	

Steps for Validation

Test Steps	**Test Results**
Ping the gateway address from SVI-NET-X-VM1	Ping successful
Check established neighbors using below command on SVI-NET-X-VM1 birdc show protocol	Neighbor should be established at least to one of the leaf nodes
Check BGP routes advertised as well as received on both SVI-NET-X-VM1 using below command birdc show route	The prefixes advertised as well as received should be shown in this output
Ping 11.10.X.1-5 from SVI-NET-XVM1	Ping should be successful

VM BGP peering with L3OUT

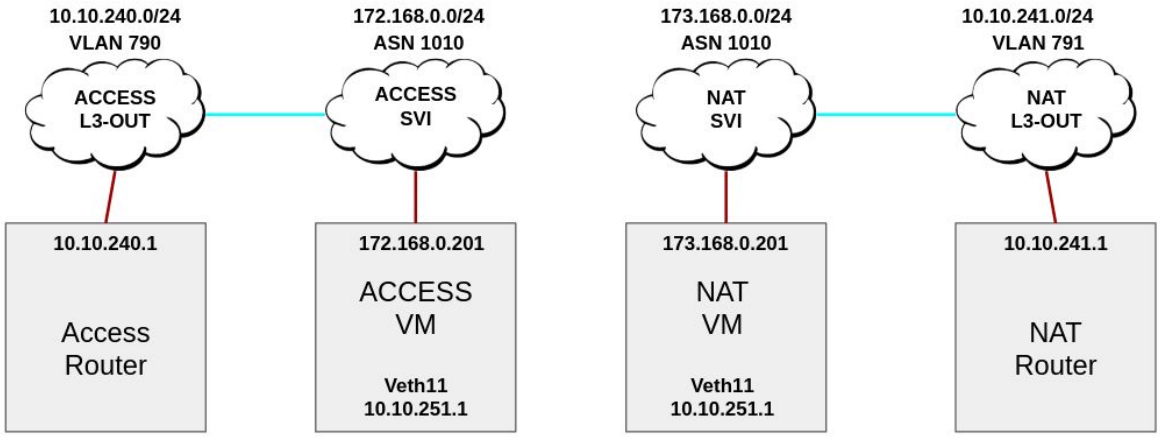
L3Outs are the Cisco Application Centric Infrastructure (ACI) objects used to provide external connectivity in external Layer 3 networks. The L3Out is where you configure the interfaces, protocols, and protocol parameters that are used to provide IP connectivity to external routers.

In this use case a VM running BGP routing stack is peering with an SVI that is connected to 3OUT.

Setup Details

In this setup we have two SVI networks. Each SVI network has a VM running bird BGP routing stack. The SVIs are also connected to two different L3OUTs. The L3OUTs are preconfigured with external routers. The connectivity to the external routers is through static routes. Even the connectivity from external routers to ACI is also static. This is also preconfigured.

Setup Diagram



Functional Test Cases

Base Topology and Configurations

Access VM Configuration

This configuration needs to be applied on the access VM for simulating a network that can be advertised to Cisco ACI and reached from outside world

```
ip link add veth11 type veth peer name veth12
ip addr add 10.10.251.1/30 dev veth11
ip link set veth11 up
```

Internet VM Configuration

This configuration needs to be applied on the internet VM for simulating a network that can be advertised to Cisco ACI and reached from outside world

```
ip link add veth11 type veth peer name veth12
ip addr add 10.10.251.1/30 dev veth11
ip link set veth11 up
```

Bird Configuration on Access VM

Bird configuration to be used for BGP peering from Access VM to Cisco ACI

```
router id 199.199.199.201;
#protocol kernel {
#       persist;              # Don't remove routes on bird shutdown
#       scan time 20;           # Scan kernel routing table every 20 seconds
#       export all;           # Default is export none
#}

# This pseudo-protocol watches all interface up/down events.
protocol device {
        scan time 10;           # Scan interfaces every 10 seconds
}

protocol static {
        ipv4;
        route 10.10.251.0/24 via 172.168.0.201;
}

protocol bgp {
        description "My BGP uplink";
        local as 1010;
        neighbor 172.168.0.199 as 1;
        ipv4 {
                import all;
                export all;
        };
        source address 172.168.0.201;     # What local address we use for the TCP connection
}

protocol bgp {
        description "My BGP uplink";
        local as 1010;
        neighbor 172.168.0.200 as 1;
        ipv4 {
                import all;
                export all;
        };
        source address 172.168.0.201;     # What local address we use for the TCP connection
}
```

Bird Configuration on Internet VM

Bird configuration to be used for BGP peering from Access VM to Cisco ACI

```
router id 199.199.199.202;
#protocol kernel {
#       persist;              # Don't remove routes on bird shutdown
#       scan time 20;         # Scan kernel routing table every 20 seconds
#       export all;           # Default is export none
#}

# This pseudo-protocol watches all interface up/down events.
protocol device {
       scan time 10;         # Scan interfaces every 10 seconds
}

protocol static {
       ipv4;
       route 10.10.251.0/24 via 173.168.0.201;
}

protocol bgp {
       description "My BGP uplink";
       local as 1020;
       neighbor 173.168.0.199 as 1;
         ipv4 {
              import all;
              export all;
       };
       source address 173.168.0.201;     # What local address we use for the TCP connection
}

protocol bgp {
       description "My BGP uplink";
       local as 1020;
       neighbor 173.168.0.200 as 1;
        ipv4 {
              import all;
              export all;
       };
       source address 173.168.0.201;     # What local address we use for the TCP connection
}
```

Steps to build base Topology

Test Steps	Test Results
Create a port on the access network	Port created successfully
Create VM with port from access network	ACCESS-VM VM created successfully
Ping the gateway address from ACCESS-VM	Ping successful
Create a port on the internet network	Port created successfully
Create VM with port from internet network	INTERNET-VM VM created successfully
Ping the gateway address from INTERNET-VM	Ping successful
Apply config to ACCESS-VM	Configuration applied
Apply config to INTERNET-VM	Configuration applied

Reachability of SVI networks from external routers

In this test case, we check the connectivity of the newly spun VMs from external routers. The ACCESS VM should be reachable from Access Router and the NAT VM should be reachable from NAT Router

Test Steps	Test Results
Ping the IP address of ACCESS-VM from access-router	Ping successful
Ping the IP address of INTERNET-VM from internet-router	Ping successful

BGP connectivity between VMs and ACI

In this test case we configure bird BGP routing stack to peer with ACI and advertise a subnet from the ACCESS and NAT VMs. We check the peer establishment as well as router advertisement.

Test Steps	Test Results
Copy bird configuration file of ACCESS-VM to ACCESS-VM	Copy successful

Copy bird configuration file of INTERNET-VM to INTERNET-VM	Copy successful
Start bird application on ACCESS-VM with below command bird -c bird-eth1.conf	Bird application started successfully
Start bird application on INTERNET-VM with below command bird -c bird-eth1.conf	Bird application started successfully
Check established neighbors using below command on ACCESS-VM birdc show protocol	Neighbor should be established at least to one of the leaf nodes
Check established neighbors using below command on INTERNET-VM birdc show protocol	Neighbor should be established at least to one of the leaf nodes
Check BGP routes advertised from ACCESS-VM using below command birdc show route	The prefixes advertised should be shown in this output
Check BGP routes advertised from INTERNET-VM using below command birdc show route	The prefixes advertised should be shown in this output

Reachability of BGP advertised route from external routers

In this test case we check the reachability of the advertised prefix from external routers. We also validate the reachability in the reverse path.

Test Steps	**Test Results**
Ping the IP address of veth11 on ACCESS-VM 10.10.251.1 from access-router	Ping successful
Ping the IP address of veth11 on INTERNET-VM 10.10.251.1 from access-router	Ping successful
Ping the access-router from ACCESS-VM with source IP address as that of veth11	Ping successful
Ping the access-router from INTERNET-VM with source IP address as that of veth11	Ping successful

Scale Test Cases

We will not have any scalability test cases for this as we have only two predefined L3-Outs we need to work with on the give Cisco ACI setup.

vCPE Use Cases

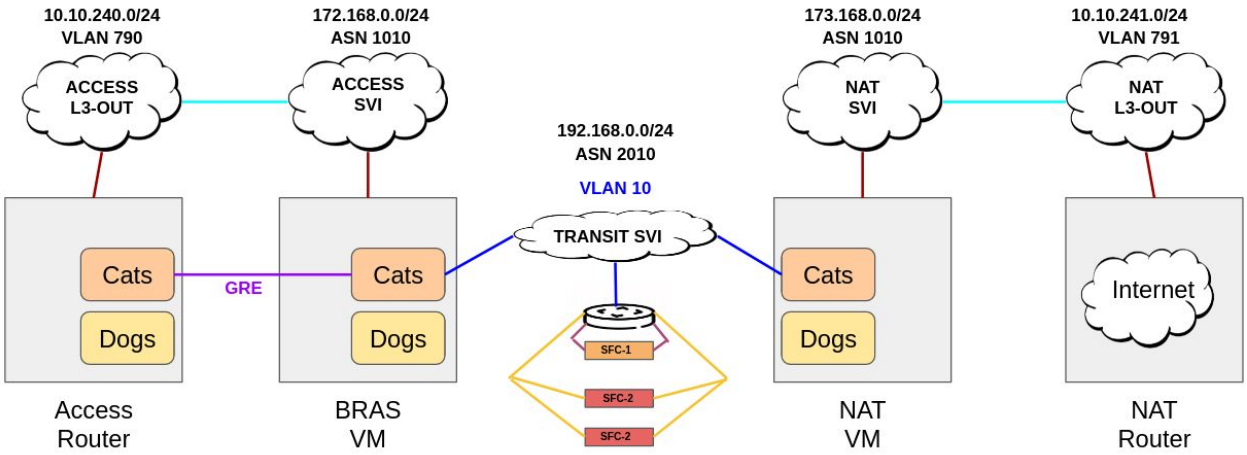
Virtual customer premises equipment (vCPE) is a way to deliver network services such as routing, firewall security and virtual private network connectivity to enterprises by using software rather than dedicated hardware devices. By virtualizing CPE, providers can dramatically simplify and accelerate service delivery, remotely configuring and managing devices and allowing customers to order new services or adjust existing ones on demand

In this use case the physical CPE at customer location is bare minimal and can route all the traffic to service provider. All the services that the customer needs are hosted in the service provider data center. We use openstack service function chaining in Cisco ACI fabric to accomplish the same.

Setup Details

This setup has Openstack deployed in Cisco ACI environment. We have an access router simulating customer traffic and NAT router simulating the internet. These two VMs are outside the openstack deployment. We terminate the customer traffic on BRAS VM. We create a transit network to pass the customer traffic to NAT VM where the traffic gets NAT before it gets to the internet. The vCPE functionality is provided in this transit network where based on customer needs we can spin service functions using openstack. By creating a transit network per customer we can provide vCPE solution per customer in provider data center.

Setup Diagram



Functional Test Cases

Base Topology and Configurations

Access Router Configuration

```
#
set -x
access_site() {
   # site 1 pete2 2 10.2.1.1 10.10.0.0/16
   inst=$1
   name=$2
   peer=$3
   addr=$4
   route=$5

   ip link add veth${inst}1 type veth peer name veth${inst}2
   ip addr add 10.10.231.${inst}/30 dev veth${inst}1
   ip link set veth${inst}1 up

   ip netns add ${name}
   ip link set veth${inst}2 netns ${name}
   ip netns exec ${name} ip addr add 10.10.231.${peer}/30 dev veth${inst}2
   ip netns exec ${name} ip link set veth${inst}2 up

   ip netns exec ${name} ip tunnel add gre${inst} mode gre \
     local 10.10.231.${peer} remote 10.10.251.${peer} dev veth${inst}2
   ip netns exec ${name} ip addr add ${addr}/24 dev gre${inst}
   ip netns exec ${name} ip link set gre${inst} up

   ip netns exec ${name} ip route add 10.10.251.0/24 via 10.10.231.${inst}
   ip netns exec ${name} ip route add ${route} dev gre${inst}

   ip netns exec ${name} ip addr add 10.0.2.1/24 dev gre${inst}
}

mksites() {
   sysctl -w net.ipv4.ip_forward=1
   access_site 1 cats 2 10.0.1.1 default
   route add -net 172.168.0.0/24 gw 10.10.240.2
   route add -net 10.10.251.0/24 gw 10.10.240.2
}
```

```
mksites "$@"
```

BRAS VM Configuration

```
#
set -x

gen_bird_conf(){
   myip=`ifconfig eth0 | grep "inet " | awk -F'[: ]+' '{ print $4 }'`
   cat /etc/bird/bird-eth0.tmpl | sed -e "s/_MY_IP_TMPL/$myip/" >/etc/bird/bird-eth0.conf
}

gen_customer_bird_conf(){
   #gen_customer_bird_conf cats 10 2010
   name=$1
   vlanid=$2
   asn=$3

   myip=`ip netns exec $name ifconfig eth0.$vlanid | grep "inet " | awk -F'[: ]+' '{ print $4 }'`
   cat /etc/bird/bird-cats.tmpl | sed -e "s/_MY_IP_TMPL/$myip/" | sed -e "s/2010/$asn/"
>/etc/bird/bird-$name.conf
}

site() {
   # site 1 pete2 2 10.2.1.1 10.10.0.0/16
   inst=$1
   name=$2
   peer=$3
   addr=$4
   route=$5

   ip link add veth${inst}1 type veth peer name veth${inst}2
   ip addr add 10.10.251.${inst}/30 dev veth${inst}1
   ip link set veth${inst}1 up

   ip netns add ${name}
   ip link set veth${inst}2 netns ${name}
   ip netns exec ${name} ip addr add 10.10.251.${peer}/30 dev veth${inst}2
   ip netns exec ${name} ip link set veth${inst}2 up

   ip netns exec ${name} ip tunnel add gre${inst} mode gre \
     local 10.10.251.${peer} remote 10.10.231.${peer} dev veth${inst}2
```

```
    ip netns exec ${name} ip addr add ${addr}/24 dev gre${inst}
    ip netns exec ${name} ip link set gre${inst} up

    ip netns exec ${name} ip route add 10.10.231.0/24 via 10.10.251.${inst}
    ip netns exec ${name} ip route add ${route} dev gre${inst}
}

mkbras() {
    sysctl -w net.ipv4.ip_forward=1
    site 1 cats 2 10.1.1.1 10.0.0.0/16
    gen_bird_conf

    ip link add link eth0 name eth0.10 type vlan id 10
    ip link set eth0.10 netns cats
    ip netns exec cats ifconfig eth0.10 hw ether fa:16:3e:bc:d5:38
    ip netns exec cats /sbin/dhclient -1 eth0.10
    sleep 4
    gen_customer_bird_conf cats 10 2010
}

# Execute from here
mkbras "$@"
```

NAT VM Configuration

```
#
set -x

gen_bird_conf(){
    myip=`ifconfig eth0 | grep "inet " | awk -F'[: ]+' '{ print $4 }'`
    cat /etc/bird/bird-eth0.tmpl | sed -e "s/_MY_IP_TMPL/$myip/" >/etc/bird/bird-eth0.conf
}

gen_customer_bird_conf(){
    #gen_customer_bird_conf cats 10 2010
    name=$1
    vlanid=$2
    asn=$3

    myip=`ip netns exec $name ifconfig eth0.$vlanid | grep "inet " | awk -F'[: ]+' '{ print $4 }'`
    cat /etc/bird/bird-cats.tmpl | sed -e "s/_MY_IP_TMPL/$myip/" | sed -e "s/2010/$asn/"
>/etc/bird/bird-$name.conf
}
```

```
nat() {
   # nat 1 pete1 2
   inst=$1
   name=$2
   peer=$3

   ip link add veth${inst}1 type veth peer name veth${inst}2
   ip addr add 10.10.251.${inst}/30 dev veth${inst}1
   ip link set veth${inst}1 up

   ip netns add ${name}
   ip link set veth${inst}2 netns ${name}
   ip netns exec ${name} ip addr add 10.10.251.${peer}/30 dev veth${inst}2
   ip netns exec ${name} ip link set veth${inst}2 up
   ip netns exec ${name} ip route add default via 10.10.251.${inst}
   ip netns exec ${name} iptables -t nat -A POSTROUTING -o veth${inst}2 -j MASQUERADE
}

mknat() {
   sysctl -w net.ipv4.ip_forward=1
   nat 1 cats 2

   gen_bird_conf
   ip link add link eth0 name eth0.10 type vlan id 10
   ip link set eth0.10 netns cats
   ip netns exec cats ifconfig eth0.10 hw ether fa:16:3e:1b:a1:a1
   ip netns exec cats /sbin/dhclient -1 eth0.10
   sleep 4
   gen_customer_bird_conf cats 10 2010
}

# Execute from here
mknat "$@"
```

NAT Router Configuration

```
#
set -x

mkinet() {
```

```
    sysctl -w net.ipv4.ip_forward=1
    ip addr add 8.8.8.1/24 dev eth0
    ip addr add 8.8.8.2/24 dev eth0
    ip addr add 8.8.8.3/24 dev eth0
    ip addr add 8.8.8.3/24 dev eth0
    ip addr add 8.8.8.4/24 dev eth0
    ip addr add 8.8.8.5/24 dev eth0
    ip addr add 8.8.8.6/24 dev eth0
    ip addr add 8.8.8.7/24 dev eth0
    ip addr add 8.8.8.8/24 dev eth0
    route add -net 10.10.251.0/24 gw 10.10.241.2
    route add -net 173.168.0.0/24 gw 10.10.241.2
}

# Execute from here
mkinet "$@"
```

Bird Command Execution

```
bird -c /etc/bird/bird-eth0.conf
ip netns exec cats bird -c /etc/bird/bird-cats.conf -P /tmp/bird-cats.run -s /tmp/sock-cats
```

Steps to build base Topology

Test Steps	**Test Results**
Create access network with following attributes --provider:network_type vlan --shared --apic:svi True --apic:bgp_enable True --apic:bgp_asn 1010 --apic:distinguished_names type=dict ExternalNetwork=uni/tn-comment/out-Access-Out/instP-data_ext_pol	The network should be successfully created without any errors
Create subnet for access network with following attributes --gateway 172.168.0.1 --subnet-range 172.168.0.0/24 --host-route destination=10.108.1.0/24,gateway=172.168.0.1 --host-route	The subnet should be created successfully without any errors

destination=10.10.240.0/24,gateway=172.168.0.1 --host-route destination=10.10.224.0/22,gateway=172.168.0.1	
Create SVI ports on all leaf switches with the following attributes. Change the IP address for each node. --device-owner apic:svi --fixed-ip subnet=”subnet-id”,ip-address=172.168.0.199	The SVI ports for all the leaf switches should be created successfully without any errors
Ping default gateway from access router	Ping should be successful
Ping SVI port IP address from access router	Ping should be successful
Create internet network with following attributes --provider:network_type vlan --shared --apic:svi True --apic:bgp_enable True --apic:bgp_asn 1020 --apic:distinguished_names type=dict ExternalNetwork=uni/tn-common/out-Internet-Out/instP-data_ext_pol	The network should be successfully created without any errors
Create subnet for internet network with following attributes --gateway 173.168.0.1 --subnet-range 173.168.0.0/24 --host-route destination=10.108.1.0/24,gateway=173.168.0.1 --host-route destination=10.10.241.0/24,gateway=173.168.0.1	The subnet should be created successfully without any errors
Create SVI ports on all leaf switches with the following attributes. Change the IP address for each node. --device-owner apic:svi --fixed-ip subnet=”subnet-id”,ip-address=173.168.0.199	The SVI ports for all the leaf switches should be created successfully without any errors
Ping default gateway from nat router	Ping should be successful
Ping SVI port IP address from nat router	Ping should be successful

Create a port on the access network	Port created successfully
Create a trunk using port created on access network	Trunk created successfully
Create VM with port from access network	BRAS1 VM created successfully
Ping the gateway address from the BRAS1 VM	Ping successful
Create a port on the internet network	Port created successfully
Create a trunk using port created on internet network	Trunk created successfully
Create VM with port from internet network	NAT VM created successfully
Ping the gateway address from the NAT VM	Ping successful
Create a project called Cats	Cats project created successfully
Create all the below resources in Cats Project	Resources being created in Cats project
Create an openstack router	Openstack router created successfully.
Create transit network with following attributes --provider:network_type vlan --apic:svi True --apic:bgp_enable True --apic:bgp_asn 2010	The network should be successfully created without any errors
Create subnet for transit network with following attributes --gateway 192.168.0.1 --subnet-range 192.168.0.0/24	The subnet should be created successfully without any errors
Create SVI ports on all leaf switches with the following attributes. Change the IP address for each node. --device-owner apic:svi --fixed-ip subnet=”subnet-id”,ip-address=192.168.0.199	The SVI ports for all the leaf switches should be created successfully without any errors
Add transit subnet to the openstack router	Subnet added successfully
Apply the configuration on Access Router	Configuration applied Successfully
Apply the configuration on BRAS1 VM	Configuration applied Successfully
Apply the configuration on NAT VM	Configuration applied Successfully

Apply the configuration on NAT Router	Configuration applied Successfully
Run bird commands on BRAS1 VM	Bird commands executed Successfully
Run bird commands on NAT VM	Bird commands executed Successfully

Single customer, single site, single sfc

In this use case we deploy the vCPE solution with a single service function chain. We would route the traffic from the access router all the way to the internet router through this sfc vm. Block some traffic and forward the rest based on sfc vm configuration.

Test Steps	**Test Results**
Ping GRE Tunnel IP address on BRAS1 cats namespace (10.1.1.1) from cats namespace on access router	Ping should be successful
Ping multiple IP address in range (8.8.8.1-8) from cats namespace on access router	Ping should be successful
Create LEFT-1 network	LEFT-1 network created successfully
Create LEFT-1 subnet with following attributes --gateway 1.1.0.1 --subnet-range 1.1.0.0/24 --host-route destination=10.0.0.0/16,gateway=1.1.0.1	LEFT-1 subnet created successfully
Add LEFT-1 subnet to the openstack router	Subnet added successfully
Create RIGHT-1 network	RIGHT-1 network created successfully
Create RIGHT-1 subnet with following attributes --gateway 2.1.0.1 --subnet-range 2.1.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.1.0.1 --host-route destination=128.0.0.0/1,gateway=2.1.0.1	RIGHT-1 subnet created successfully
Add RIGHT-1 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-1 network	Port created successfully
Create a port on RIGHT-1 network	Port created successfully
Create a Service VM with ports from LEFT-1	Service VM created successfully

and RIGHT-1 networks with OpenWRTSI1 image	
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 10.0.1.0/24 --l7-parameters logical_source_network=TRANSIT-NET-ID, logical_destination_network=TRANSIT-NET-ID	Flow Classifier created successfully
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully
Ping GRE Tunnel IP address on BRAS1 cats namespace (10.1.1.1) from cats namespace on access router	Ping should be successful
Ping multiple IP address in range (8.8.8.1-8) from cats namespace on access router	Ping should be successful
Create a port chain with the flow classifier and port pair group	Service Function Chain created successfully
Ping multiple IP address in range (8.8.8.1-8) from cats namespace on access router	Ping successful and going through the newly created SFC except 8.8.8.1

Single customer, single site, multiple sfcs

In this use case we deploy the vCPE solution with multiple service function vms in a chain. We cancade traffic from one service to another until it reaches the nat router. Based on the functionality defined in the service vms in the service chain traffic will be allowed or denied for a certain set of IP address.

Test Steps	**Test Results**
Ping GRE Tunnel IP address on BRAS1 cats namespace (10.1.1.1) from cats namespace on access router	Ping should be successful
Ping multiple IP address in range (8.8.8.1-8) from cats namespace on access router	Ping should be successful

Create LEFT-1 network	LEFT-1 network created successfully
Create LEFT-1 subnet with following attributes --gateway 1.1.0.1 --subnet-range 1.1.0.0/24 --host-route destination=10.0.0.0/16,gateway=1.1.0.1	LEFT-1 subnet created successfully
Add LEFT-1 subnet to the openstack router	Subnet added successfully
Create RIGHT-1 network	RIGHT-1 network created successfully
Create RIGHT-1 subnet with following attributes --gateway 2.1.0.1 --subnet-range 2.1.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.1.0.1 --host-route destination=128.0.0.0/1,gateway=2.1.0.1	RIGHT-1 subnet created successfully
Add RIGHT-1 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-1 network	Port created successfully
Create a port on RIGHT-1 network	Port created successfully
Create a Service VM with ports from LEFT-1 and RIGHT-1 networks with OpenWRTSI1 image	Service VM created successfully
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 10.0.1.0/24 --l7-parameters logical_source_network=TRANSIT-NET-ID, logical_destination_network=TRANSIT-NET-ID	Flow Classifier created successfully
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully
Ping GRE Tunnel IP address on BRAS1 cats namespace (10.1.1.1) from cats namespace on access router	Ping should be successful
Ping multiple IP address in range (8.8.8.1-8) from cats namespace on access router	Ping should be successful

Create a port chain with the flow classifier and port pair group	Service Function Chain created successfully
Ping multiple IP address in range (8.8.8.1-8) from cats namespace on access router	Ping successful and going through the newly created SFC except 8.8.8.1
Create LEFT-2 network	LEFT-2 network created successfully
Create LEFT-2 subnet with following attributes --gateway 1.2.0.1 --subnet-range 1.2.0.0/24 --host-route destination=10.0.0.0/16,gateway=1.2.0.1	LEFT-2 subnet created successfully
Add LEFT-2 subnet to the openstack router	Subnet added successfully
Create RIGHT-2 network	RIGHT-2 network created successfully
Create RIGHT-2 subnet with following attributes --gateway 2.2.0.1 --subnet-range 2.2.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.2.0.1 --host-route destination=128.0.0.0/1,gateway=2.2.0.1	RIGHT-2 subnet created successfully
Add RIGHT-2 subnet to the openstack router	Subnet added successfully
Create three port on LEFT-2 network	Ports created successfully
Create three port on RIGHT-2 network	Ports created successfully
Create three Service VM with ports from LEFT-2 and RIGHT-2 networks with OpenWRTSI2 image	Service VMs created successfully
Create three port pair with ports used for creating the service VMs	Port Pairs create successfully
Create a port pair group with the port pairs created	Port Pair Group create successfully
Ping multiple IP address in range (8.8.8.1-8) from cats namespace on access router	Ping successful and going through the newly created SFC except 8.8.8.1
Update the service function chain with the newly created port pair group	Service Function Chain updated successfully
Ping multiple IP address in range (8.8.8.1-8)	All Ping successful except 8.8.8.1, 8.8.8.2.

from cats namespace on access router	Blocked by Service VMs
Create LEFT-3 network	LEFT-3 network created successfully
Create LEFT-3 subnet with following attributes --gateway 1.3.0.1 --subnet-range 1.3.0.0/24 --host-route destination=10.0.0.0/16,gateway=1.3.0.1	LEFT-3 subnet created successfully
Add LEFT-3 subnet to the openstack router	Subnet added successfully
Create RIGHT-3 network	RIGHT-3 network created successfully
Create RIGHT-3 subnet with following attributes --gateway 2.3.0.1 --subnet-range 2.3.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.3.0.1 --host-route destination=128.0.0.0/1,gateway=2.3.0.1	RIGHT-3 subnet created successfully
Add RIGHT-3 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-3 network	Port created successfully
Create a port on RIGHT-3 network	Port created successfully
Create a Service VM with ports from LEFT-3 and RIGHT-3 networks with OpenWRTSI3 image	Service VM created successfully
Create a port pair with ports used for creating the service VMs	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully
Ping multiple IP address in range (8.8.8.1-8) from cats namespace on access router	All Ping successful except 8.8.8.1, 8.8.8.2. Blocked by Service VMs
Update the service function chain with the newly created port pair group	Service Function Chain updated successfully
Ping multiple IP address in range (8.8.8.1-8) from cats namespace on access router	All Ping successful except 8.8.8.1, 8.8.8.2 and 8.8.8.3 Blocked by Service VMs

Single customer, two sites, single sfc

In this use case instead of one site, we simulate two sites for the same customer. This is accomplished by terminating each site on a different BRAS VM. We would route all of this customer traffic through the same vCPE service chain even when traffic is coming from two different sites. This is like having a unified policy for all the traffic coming from various sites to have the same treatment across the vCPE solution.

Access Router Configuration

```
#

set -x

access_site() {
   # site 1 pete2 2 10.2.1.1 10.10.0.0/16
   inst=$1
   name=$2
   peer=$3
   addr=$4
   route=$5

   ip link add veth${inst}1 type veth peer name veth${inst}2
   ip addr add 10.10.231.${inst}/30 dev veth${inst}1
   ip link set veth${inst}1 up

   ip netns add ${name}
   ip link set veth${inst}2 netns ${name}
   ip netns exec ${name} ip addr add 10.10.231.${peer}/30 dev veth${inst}2
   ip netns exec ${name} ip link set veth${inst}2 up

   ip netns exec ${name} ip tunnel add gre${inst} mode gre \
     local 10.10.231.${peer} remote 10.10.251.${peer} dev veth${inst}2
   ip netns exec ${name} ip addr add ${addr}/24 dev gre${inst}
   ip netns exec ${name} ip link set gre${inst} up

   ip netns exec ${name} ip route add 10.10.251.0/24 via 10.10.231.${inst}
   ip netns exec ${name} ip route add ${route} dev gre${inst}
}

mksites() {
   sysctl -w net.ipv4.ip_forward=1
```

```
    access_site 1 cats-site1 2 10.0.1.1 default
    access_site 5 cats-site2 6 10.0.2.1 default
    route add -net 172.168.0.0/24 gw 10.10.240.2
    route add -net 10.10.251.0/24 gw 10.10.240.2
}

mksites "$@"
```

BRAS1 VM Configuration

```
#

set -x

gen_bird_conf(){
    myip=`ifconfig eth0 | grep "inet " | awk -F'[: ]+' '{ print $4 }'`
    cat /etc/bird/bird-eth0.tmpl | sed -e "s/_MY_IP_TMPL/$myip/" | sed -e
"s/10.10.251.0\/24/10.10.251.0\/30/" >/etc/bird/bird-eth0.conf
}

gen_customer_bird_conf(){
    #gen_customer_bird_conf cats 10 2010
    name=$1
    vlanid=$2
    asn=$3

    myip=`ip netns exec $name ifconfig eth0.$vlanid | grep "inet " | awk -F'[: ]+' '{ print $4 }'`
    cat /etc/bird/bird-cats.tmpl | sed -e "s/_MY_IP_TMPL/$myip/" | sed -e "s/2010/$asn/"
>/etc/bird/bird-$name.conf
}

site() {
    # site 1 pete2 2 10.2.1.1 10.10.0.0/16
    inst=$1
    name=$2
    peer=$3
    addr=$4
    route=$5

    ip link add veth${inst}1 type veth peer name veth${inst}2
    ip addr add 10.10.251.${inst}/30 dev veth${inst}1
    ip link set veth${inst}1 up
```

```
    ip netns add ${name}
    ip link set veth${inst}2 netns ${name}
    ip netns exec ${name} ip addr add 10.10.251.${peer}/30 dev veth${inst}2
    ip netns exec ${name} ip link set veth${inst}2 up

    ip netns exec ${name} ip tunnel add gre${inst} mode gre \
      local 10.10.251.${peer} remote 10.10.231.${peer} dev veth${inst}2
    ip netns exec ${name} ip addr add ${addr}/24 dev gre${inst}
    ip netns exec ${name} ip link set gre${inst} up

    ip netns exec ${name} ip route add 10.10.231.0/24 via 10.10.251.${inst}
    ip netns exec ${name} ip route add ${route} dev gre${inst}
}

mkbras() {
    sysctl -w net.ipv4.ip_forward=1
    site 1 cats 2 10.1.1.1 10.0.0.0/16
    gen_bird_conf

    ip link add link eth0 name eth0.10 type vlan id 10
    ip link set eth0.10 netns cats
    ip netns exec cats ifconfig eth0.10 hw ether fa:16:3e:bc:d5:38
    ip netns exec cats /sbin/dhclient -1 eth0.10
    sleep 4
    gen_customer_bird_conf cats 10 2010
}

# Execute from here
mkbras "$@"
```

BRAS2 VM Configuration

```
#

set -x

gen_bird_conf(){
    myip=`ifconfig eth0 | grep "inet " | awk -F'[: ]+' '{ print $4 }'`
    cat /etc/bird/bird-eth0.tmpl | sed -e "s/_MY_IP_TMPL/$myip/" | sed -e
"s/10.10.251.0\/24/10.10.251.4\/30/" >/etc/bird/bird-eth0.conf
}
```

```
gen_customer_bird_conf(){
   #gen_customer_bird_conf cats 10 2010
   name=$1
   vlanid=$2
   asn=$3

   myip=`ip netns exec $name ifconfig eth0.$vlanid | grep "inet " | awk -F'[: ]+' '{ print $4 }'`
   cat /etc/bird/bird-cats.tmpl | sed -e "s/_MY_IP_TMPL/$myip/" | sed -e "s/2010/$asn/"| sed -e
"s/10.0.1.0/10.0.2.0/" >/etc/bird/bird-$name.conf
}

site() {
   # site 1 pete2 2 10.2.1.1 10.10.0.0/16
   inst=$1
   name=$2
   peer=$3
   addr=$4
   route=$5

   ip link add veth${inst}1 type veth peer name veth${inst}2
   ip addr add 10.10.251.${inst}/30 dev veth${inst}1
   ip link set veth${inst}1 up

   ip netns add ${name}
   ip link set veth${inst}2 netns ${name}
   ip netns exec ${name} ip addr add 10.10.251.${peer}/30 dev veth${inst}2
   ip netns exec ${name} ip link set veth${inst}2 up

   ip netns exec ${name} ip tunnel add gre${inst} mode gre \
     local 10.10.251.${peer} remote 10.10.231.${peer} dev veth${inst}2
   ip netns exec ${name} ip addr add ${addr}/24 dev gre${inst}
   ip netns exec ${name} ip link set gre${inst} up

   ip netns exec ${name} ip route add 10.10.231.0/24 via 10.10.251.${inst}
   ip netns exec ${name} ip route add ${route} dev gre${inst}
}

mkbras() {
   sysctl -w net.ipv4.ip_forward=1
   site 5 cats 6 10.1.1.1 10.0.0.0/16
   gen_bird_conf
```

```
    ip link add link eth0 name eth0.10 type vlan id 10
    ip link set eth0.10 netns cats
    ip netns exec cats ifconfig eth0.10 hw ether fa:16:3e:bc:d5:37
    ip netns exec cats /sbin/dhclient -1 eth0.10
    sleep 4
    gen_customer_bird_conf cats 10 2010
}

# Execute from here
mkbras "$@"
```

NAT VM Configuration

```
#

set -x

gen_bird_conf(){
    myip=`ifconfig eth0 | grep "inet " | awk -F'[: ]+' '{ print $4 }'`
    cat /etc/bird/bird-eth0.tmpl | sed -e "s/_MY_IP_TMPL/$myip/" >/etc/bird/bird-eth0.conf
}

gen_customer_bird_conf(){
    #gen_customer_bird_conf cats 10 2010
    name=$1
    vlanid=$2
    asn=$3

    myip=`ip netns exec $name ifconfig eth0.$vlanid | grep "inet " | awk -F'[: ]+' '{ print $4 }'`
    cat /etc/bird/bird-cats.tmpl | sed -e "s/_MY_IP_TMPL/$myip/" | sed -e "s/2010/$asn/" | sed -e
"s/10.0.1.0\/24/10.0.0.0\/22/" >/etc/bird/bird-$name.conf
}

nat() {
    # nat 1 pete1 2
    inst=$1
    name=$2
    peer=$3

    ip link add veth${inst}1 type veth peer name veth${inst}2
    ip addr add 10.10.251.${inst}/30 dev veth${inst}1
    ip link set veth${inst}1 up
```

```
    ip netns add ${name}
    ip link set veth${inst}2 netns ${name}
    ip netns exec ${name} ip addr add 10.10.251.${peer}/30 dev veth${inst}2
    ip netns exec ${name} ip link set veth${inst}2 up
    ip netns exec ${name} ip route add default via 10.10.251.${inst}
    ip netns exec ${name} iptables -t nat -A POSTROUTING -o veth${inst}2 -j MASQUERADE
}

mknat() {
    sysctl -w net.ipv4.ip_forward=1
    nat 1 cats 2

    gen_bird_conf
    ip link add link eth0 name eth0.10 type vlan id 10
    ip link set eth0.10 netns cats
    ip netns exec cats ifconfig eth0.10 hw ether fa:16:3e:1b:a1:a1
    ip netns exec cats /sbin/dhclient -1 eth0.10
    sleep 4
    gen_customer_bird_conf cats 10 2010
}

# Execute from here
mknat "$@"
```

Test Steps	**Test Results**
Ping GRE Tunnel IP address on BRAS1 cats namespace (10.1.1.1) from cats-site1 namespace on access router	Ping should be successful
Ping multiple IP address in range (8.8.8.1-8) from cats-site1 namespace on access router	Ping should be successful
Ping GRE Tunnel IP address on BRAS2 cats namespace (10.1.1.1) from cats-site2 namespace on access router	Ping should be successful
Ping multiple IP address in range (8.8.8.1-8) from cats-site2 namespace on access router	Ping should be successful
Create LEFT-1 network	LEFT-1 network created successfully
Create LEFT-1 subnet with following attributes	LEFT-1 subnet created successfully

--gateway 1.1.0.1 --subnet-range 1.1.0.0/24 --host-route destination=10.0.0.0/16,gateway=1.1.0.1	
Add LEFT-1 subnet to the openstack router	Subnet added successfully
Create RIGHT-1 network	RIGHT-1 network created successfully
Create RIGHT-1 subnet with following attributes --gateway 2.1.0.1 --subnet-range 2.1.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.1.0.1 --host-route destination=128.0.0.0/1,gateway=2.1.0.1	RIGHT-1 subnet created successfully
Add RIGHT-1 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-1 network	Port created successfully
Create a port on RIGHT-1 network	Port created successfully
Create a Service VM with ports from LEFT-1 and RIGHT-1 networks with OpenWRTSI1 image	Service VM created successfully
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 10.0.1.0/24 --l7-parameters logical_source_network=TRANSIT-NET-ID, logical_destination_network=TRANSIT-NET-ID	Flow Classifier created successfully
Create one more flow classifier with following attributes for second site --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 10.0.2.0/24 --l7-parameters logical_source_network=TRANSIT-NET-ID, logical_destination_network=TRANSIT-NET-ID	Flow Classifier created successfully
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully
Ping GRE Tunnel IP address on BRAS1 cats	Ping should be successful

namespace (10.1.1.1) from cats-site1 namespace on access router	
Ping multiple IP address in range (8.8.8.1-8) from cats-site1 namespace on access router	Ping should be successful
Ping GRE Tunnel IP address on BRAS2 cats namespace (10.1.1.1) from cats-site2 namespace on access router	Ping should be successful
Ping multiple IP address in range (8.8.8.1-8) from cats-site2 namespace on access router	Ping should be successful
Create a port chain with the flow classifiers and port pair group	Service Function Chain created successfully
Ping multiple IP address in range (8.8.8.1-8) from cats-site1 namespace on access router	Ping successful and going through the newly created SFC except 8.8.8.1
Ping multiple IP address in range (8.8.8.1-8) from cats-site2 namespace on access router	Ping successful and going through the newly created SFC except 8.8.8.1

Two customers, single site, single sfc

In this use case we simulate two customers on the access router using name spaces. We terminate the traffic from different customers on the same BRAS VM using different name spaces. We create a transit network per customer and simulate vCPE solution per customer. We will use the same IPv4 address space for both the customers to show that we are agnostic to what IP address the customer uses. Traffic is validated through respective service VMs for allow and drop before it reaches the NAT router.

Access Router Configuration

```
#

set -x

access_site() {
   # site 1 pete2 2 10.2.1.1 10.10.0.0/16
   inst=$1
   name=$2
   peer=$3
   addr=$4
   route=$5
```

```
    ip link add veth${inst}1 type veth peer name veth${inst}2
    ip addr add 10.10.231.${inst}/30 dev veth${inst}1
    ip link set veth${inst}1 up

    ip netns add ${name}
    ip link set veth${inst}2 netns ${name}
    ip netns exec ${name} ip addr add 10.10.231.${peer}/30 dev veth${inst}2
    ip netns exec ${name} ip link set veth${inst}2 up

    ip netns exec ${name} ip tunnel add gre${inst} mode gre \
      local 10.10.231.${peer} remote 10.10.251.${peer} dev veth${inst}2
    ip netns exec ${name} ip addr add ${addr}/24 dev gre${inst}
    ip netns exec ${name} ip link set gre${inst} up

    ip netns exec ${name} ip route add 10.10.251.0/24 via 10.10.231.${inst}
    ip netns exec ${name} ip route add ${route} dev gre${inst}

    ip netns exec ${name} ip addr add 10.0.2.1/24 dev gre${inst}
}

mksites() {
    sysctl -w net.ipv4.ip_forward=1
    access_site 1 cats 2 10.0.1.1 default
    access_site 5 dogs 6 10.0.1.1 default
    route add -net 172.168.0.0/24 gw 10.10.240.2
    route add -net 10.10.251.0/24 gw 10.10.240.2
}

mksites "$@"
```

BRAS VM Configuration

```
#

set -x

gen_bird_conf(){
    myip=`ifconfig eth0 | grep "inet " | awk -F'[: ]+' '{ print $4 }'`
    cat /etc/bird/bird-eth0.tmpl | sed -e "s/_MY_IP_TMPL/$myip/" >/etc/bird/bird-eth0.conf
}

gen_customer_bird_conf(){
```

```
    #gen_customer_bird_conf cats 10 2010
    name=$1
    vlanid=$2
    asn=$3

    myip=`ip netns exec $name ifconfig eth0.$vlanid | grep "inet " | awk -F'[: ]+' '{ print $4 }'`
    cat /etc/bird/bird-cats.tmpl | sed -e "s/_MY_IP_TMPL/$myip/" | sed -e "s/2010/$asn/"
>/etc/bird/bird-$name.conf
}

site() {
    # site 1 pete2 2 10.2.1.1 10.10.0.0/16
    inst=$1
    name=$2
    peer=$3
    addr=$4
    route=$5

    ip link add veth${inst}1 type veth peer name veth${inst}2
    ip addr add 10.10.251.${inst}/30 dev veth${inst}1
    ip link set veth${inst}1 up

    ip netns add ${name}
    ip link set veth${inst}2 netns ${name}
    ip netns exec ${name} ip addr add 10.10.251.${peer}/30 dev veth${inst}2
    ip netns exec ${name} ip link set veth${inst}2 up

    ip netns exec ${name} ip tunnel add gre${inst} mode gre \
      local 10.10.251.${peer} remote 10.10.231.${peer} dev veth${inst}2
    ip netns exec ${name} ip addr add ${addr}/24 dev gre${inst}
    ip netns exec ${name} ip link set gre${inst} up

    ip netns exec ${name} ip route add 10.10.231.0/24 via 10.10.251.${inst}
    ip netns exec ${name} ip route add ${route} dev gre${inst}
}

mkbras() {
    sysctl -w net.ipv4.ip_forward=1
    site 1 cats 2 10.1.1.1 10.0.0.0/16
    site 5 dogs 6 10.1.1.1 10.0.0.0/16
    gen_bird_conf

    ip link add link eth0 name eth0.10 type vlan id 10
```

```
    ip link set eth0.10 netns cats
    ip netns exec cats ifconfig eth0.10 hw ether fa:16:3e:bc:d5:38
    ip netns exec cats /sbin/dhclient -1 eth0.10
    sleep 4
    gen_customer_bird_conf cats 10 2010

    ip link add link eth0 name eth0.20 type vlan id 20
    ip link set eth0.20 netns dogs
    ip netns exec dogs ifconfig eth0.20 hw ether fa:16:3e:bc:d5:39
    ip netns exec dogs /sbin/dhclient -1 eth0.20
    sleep 4
    gen_customer_bird_conf dogs 20 3010
}

# Execute from here
mkbras "$@"
```

NAT VM Configuration

```
#

set -x

gen_bird_conf(){
    myip=`ifconfig eth0 | grep "inet " | awk -F'[: ]+' '{ print $4 }'`
    cat /etc/bird/bird-eth0.tmpl | sed -e "s/_MY_IP_TMPL/$myip/" >/etc/bird/bird-eth0.conf
}

gen_customer_bird_conf(){
    #gen_customer_bird_conf cats 10 2010
    name=$1
    vlanid=$2
    asn=$3

    myip=`ip netns exec $name ifconfig eth0.$vlanid | grep "inet " | awk -F'[: ]+' '{ print $4 }'`
    cat /etc/bird/bird-cats.tmpl | sed -e "s/_MY_IP_TMPL/$myip/" | sed -e "s/2010/$asn/"
>/etc/bird/bird-$name.conf
}

nat() {
    # nat 1 pete1 2
    inst=$1
    name=$2
```

```
    peer=$3

    ip link add veth${inst}1 type veth peer name veth${inst}2
    ip addr add 10.10.251.${inst}/30 dev veth${inst}1
    ip link set veth${inst}1 up

    ip netns add ${name}
    ip link set veth${inst}2 netns ${name}
    ip netns exec ${name} ip addr add 10.10.251.${peer}/30 dev veth${inst}2
    ip netns exec ${name} ip link set veth${inst}2 up
    ip netns exec ${name} ip route add default via 10.10.251.${inst}
    ip netns exec ${name} iptables -t nat -A POSTROUTING -o veth${inst}2 -j MASQUERADE
}

mknat() {
    sysctl -w net.ipv4.ip_forward=1
    nat 1 cats 2
    nat 5 dogs 6

    gen_bird_conf
    ip link add link eth0 name eth0.10 type vlan id 10
    ip link set eth0.10 netns cats
    ip netns exec cats ifconfig eth0.10 hw ether fa:16:3e:1b:a1:a1
    ip netns exec cats /sbin/dhclient -1 eth0.10
    sleep 4
    gen_customer_bird_conf cats 10 2010

    ip link add link eth0 name eth0.20 type vlan id 20
    ip link set eth0.20 netns dogs
    ip netns exec dogs ifconfig eth0.20 hw ether fa:16:3e:1b:a1:a2
    ip netns exec dogs /sbin/dhclient -1 eth0.20
    sleep 4
    gen_customer_bird_conf dogs 20 3010
}

# Execute from here
mknat "$@"
```

Test Steps	**Test Results**
Create a project called Dogs	Dogs project created successfully

Create all the below resources in Dogs Project	Resources being created in Dogs project
Create an openstack router	Openstack router created successfully.
Create transit network with following attributes --provider:network_type vlan --apic:svi True --apic:bgp_enable True --apic:bgp_asn 3010	The network should be successfully created without any errors
Create subnet for transit network with following attributes --gateway 192.168.0.1 --subnet-range 192.168.0.0/24	The subnet should be created successfully without any errors
Create SVI ports on all leaf switches with the following attributes. Change the IP address for each node. --device-owner apic:svi --fixed-ip subnet=”subnet-id”,ip-address=192.168.0.199	The SVI ports for all the leaf switches should be created successfully without any errors
Add transit subnet to the openstack router	Subnet added successfully
Change project to Cats	Project change successful
Ping GRE Tunnel IP address on BRAS1 cats namespace (10.1.1.1) from cats namespace on access router	Ping should be successful
Ping multiple IP address in range (8.8.8.1-8) from cats namespace on access router	Ping should be successful
Create LEFT-1 network	LEFT-1 network created successfully
Create LEFT-1 subnet with following attributes --gateway 1.1.0.1 --subnet-range 1.1.0.0/24 --host-route destination=10.0.0.0/16,gateway=1.1.0.1	LEFT-1 subnet created successfully
Add LEFT-1 subnet to the openstack router	Subnet added successfully
Create RIGHT-1 network	RIGHT-1 network created successfully
Create RIGHT-1 subnet with following attributes --gateway 2.1.0.1 --subnet-range 2.1.0.0/24 --host-route	RIGHT-1 subnet created successfully

destination=0.0.0.0/1,gateway=2.1.0.1 --host-route destination=128.0.0.0/1,gateway=2.1.0.1	
Add RIGHT-1 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-1 network	Port created successfully
Create a port on RIGHT-1 network	Port created successfully
Create a Service VM with ports from LEFT-1 and RIGHT-1 networks with OpenWRTSI1 image	Service VM created successfully
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 10.0.1.0/24 --l7-parameters logical_source_network=TRANSIT-NET-ID, logical_destination_network=TRANSIT-NET-ID	Flow Classifier created successfully
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully
Ping GRE Tunnel IP address on BRAS1 cats namespace (10.1.1.1) from cats namespace on access router	Ping should be successful
Ping multiple IP address in range (8.8.8.1-8) from cats namespace on access router	Ping should be successful
Create a port chain with the flow classifier and port pair group	Service Function Chain created successfully
Ping multiple IP address in range (8.8.8.1-8) from cats namespace on access router	Ping successful and going through the newly created SFC except 8.8.8.1
Change project to Dogs	Project change successful
Ping GRE Tunnel IP address on BRAS1 dogs namespace (10.1.1.1) from dogs namespace on access router	Ping should be successful
Ping multiple IP address in range (8.8.8.1-8) from dogs namespace on access router	Ping should be successful

Create LEFT-1 network	LEFT-1 network created successfully
Create LEFT-1 subnet with following attributes --gateway 1.1.0.1 --subnet-range 1.1.0.0/24 --host-route destination=10.0.0.0/16,gateway=1.1.0.1	LEFT-1 subnet created successfully
Add LEFT-1 subnet to the openstack router	Subnet added successfully
Create RIGHT-1 network	RIGHT-1 network created successfully
Create RIGHT-1 subnet with following attributes --gateway 2.1.0.1 --subnet-range 2.1.0.0/24 --host-route destination=0.0.0.0/1,gateway=2.1.0.1 --host-route destination=128.0.0.0/1,gateway=2.1.0.1	RIGHT-1 subnet created successfully
Add RIGHT-1 subnet to the openstack router	Subnet added successfully
Create a port on LEFT-1 network	Port created successfully
Create a port on RIGHT-1 network	Port created successfully
Create a Service VM with ports from LEFT-1 and RIGHT-1 networks with OpenWRTSI1 image	Service VM created successfully
Create a flow classifier with following attributes --destination-ip-prefix 0.0.0.0/0 --source-ip-prefix 10.0.1.0/24 --l7-parameters logical_source_network=TRANSIT-NET-ID, logical_destination_network=TRANSIT-NET-ID	Flow Classifier created successfully
Create a port pair with ports used for creating the service VM	Port Pair create successfully
Create a port pair group with the port pair created	Port Pair Group create successfully
Ping GRE Tunnel IP address on BRAS1 cats namespace (10.1.1.1) from cats namespace on access router	Ping should be successful
Ping multiple IP address in range (8.8.8.1-8) from cats namespace on access router	Ping should be successful

Create a port chain with the flow classifier and port pair group	Service Function Chain created successfully
Ping multiple IP address in range (8.8.8.1-8) from cats namespace on access router	Ping successful and going through the newly created SFC except 8.8.8.1

Scale Test Cases

vCPE Scale Scenario

In this scale user case we will take the simplest of the functional use cases and scale on it. The scale here is more from the no. of customers we can support on Cisco ACI with a vCPE solution. The simplest of vCPE solution is with one VM for a SFC functionality. So we will create X no. of customers where the variable value can vary from 2, 5, 10 to 100. The script would create so many projects, so many namespaces to simulate customers and GRE tunnels from customer sites to the same number of Transit SVIs. The validation for this test has to happen from every customer namespace all the way to the internet.