Virtual Flow Forwarder VNF

The goal of SFCLola is to handle SFC requests within a tenant's virtual network domain spanning multiple DCs. Our reference use case consists of multiple instances of different types of VNFs deployed in virtual networks managed by the tenant and hosted at multiple and geographically distributed sites.

The problem we address is the setup of a service chain across VF instances deployed within a tenant virtual network, also spanning multiple DCs.

SFC management within the tenant domain implies that traffic routing along chained VNFs should be enforced by programmable forwarding elements that manage traffic flow routing in the tenant data plane, without relying on management capabilities at the NFVI level, pertaining to the infrastructure operator's domain (e.g. Openstack SFC API, SFC API offered by infrastructure controller).

In our solution, the forwarding action is implemented as a software component executed by a VM, which thus takes the name of Virtual Flow Forwarder (VFF). A VFF acts as a forwarding device in the overlay network, i.e., all the packets exiting from/directed to a VNF node has to be forwarded to the VFF, which decides the next hop, according to the installed forwarding rules. The VFF offers a set of REST APIs allowing to programmatically environment. The VFF offers an intent-based REST API for creating or deleting service chains or parts of service chain within a DC.

The REST API offered by the VFF has been designed as an intent-based interface, so that the client only provides functional specifications of the actions to be accomplished.

Since the chaining sequence across VNFs is enforced by the VFF, which is the only element in an intra-DC overlay network aware of service chaining, endpoint nodes and VNFs must be configured to send their traffic to the VFF. Then, the VFF will forward the packet flow to the next hop in the chain, no matter the IP destination of the end-to-end service.

The VFF uses the iptables user-space utility program to configure the firewall tables of netfilter modules. It enforces rules on traffic selection and forwarding by classifying and marking the traffic forcing the use of a specific routing table.

Connecting to the VNF machine

1. Connect to the VPN where the VNF machine is deployed
2. Connect to the jump machine over SSH.
3. From the jump machine's shell, connect to the VNF machine over SSH. Current credentials are as follows:
   username: ubuntu
   password: 5ginfire

Running an experiment

VFF Node configuration

In the VFF Node perform the following operations:

- Enable the ipv4 forward
- Disable ICMP redirects (both send and receive)

1) In /etc/iproute2/rt_tables
[Optional] Add a list of n VNFs in the network

```
1 vf1
2 vf2
512 vff2
. . .
n vfn
```

2) for each VF in the list (routing del kernel linux)

```
// Associate table with route
sudo ip route add 0.0.0.0/0 via 192.168.9.122 table 1
sudo ip route add 0.0.0.0/0 dev ipiptun2 table 512

//Link table with mark
sudo ip rule add from all fwmark 1 table 1
sudo ip rule add from all fwmark 512 table 512
```

Where 192.168.9.122 is the IP of the VF-1 and ipiptun2 the tunnel used to reach VFF-2

VNF Node Configuration

VNF instances that should be chained thorugh the VFF have to be set so that traffic flows are forwarded to the VFF node IP.

REST APIs offered by the VFF

Hereafter we provide the documentation of the REST APIs exposed by a VFF for forwarding rules installation, retrieval and deletion. APIs are exposed on port 8080.

Create a new flow

POST /chain	
Request body	Response body

`{`
`"src":"10.10.1.1",`
`"dst":"10.10.1.2",`
`"protocol":"udp",`
`"sport":9999,`
`"dport":9999,`
`"vfs":["VF-1,VFF-2","VFF-2"]`
`}`
	`{`
	`    "request": {`
	`        "protocol": "udp",`
	`        "dport": 9999,`
	`        "vfs": [...]`
	`    },`
	`    "timestamp": 1536833560674,`
	`    "uuid": "19ffad3c-…",`
	`    "cmds": […],`
	`    "delCmds": […]`
	`}`

Valid protocols: "udp" “icmp” “tcp” “ip”	

List active flows

GET /chain

```
[
    {
        "request": {
            "protocol": "udp",
            "dport": 9999,
            "vfs": [
                {
                    "mac": "00:00:00:00:00:A1",
                    "tableIndex": 1,
                    "name": "VF-1a",
                    "type": "VF-1"
                }
            ]
        },
        "timestamp": null,
        "uuid": "680af733-…",
        "cmds": […],
        "delCmds": […]
    },
    {…},
    {…}
 ]
```

Show flow by uuid

GET /chain/{uuid}
See GET /chain

Delete flow by uuid

DELETE /chain/{uuid}
204 No Content

Show topology linked to the VFF

GET /topology
Shows topology as a JSON

Example of flow installation

In this subsection we provide an example of how a chaining request is translated into a set of forwarding rules.

We consider the following example message body for a **POST /chain** request:

```
{
"src":"10.10.10.4",
"dst":"10.10.10.5",
"protocol":"udp",
"dport":9999,
"vfs":["VF-1","VF-2"]
}
```

This means that UDP traffic flows with destination port 9999 coming from Node-A (10.10.10.4) and directed to Node-B (10.10.10.5) should be steered though VF-1 (10.10.10.10 00:00:00:00:AA:10) and VF-2 (10.10.10.20 00:00:00:00:BB:20).

As a prerequisite, the following routing rules are set in the VFF node.

```
ip route add 0.0.0.0/0 via 10.10.10.10 table 1
ip route add 0.0.0.0/0 via 10.10.10.20 table 2
ip rule add from all fwmark 1 table 1
ip rule add from all fwmark 2 table 2
```

The iptables rules corresponding to the request reported above will then be like:

```
iptables -A PREROUTING -t mangle -m mac ! --mac-source 00:00:00:00:BB:20 -p udp
--dport 9999 -s 10.10.10.4 -d 10.10.10.5 -j MARK --set-mark 1

iptables -A PREROUTING -t mangle -m mac --mac-source 00:00:00:00:AA:10 -p udp
--dport 9999 -s 10.10.10.4 -d 10.10.10.5 -j MARK --set-mark 2
```

The first rule will mark the specified traffic with mark 1 and routing table 1 will be used. This means that this traffic will be steered to VF-1 as specified in table 1. This rule is not applied for traffic coming from VF-2 avoiding loops. The second rule is applied only to traffic coming from VF-1 that is sent to VF-2. Traffic going out from VF-2 follows its typical path to its final destination.