LF NETWORKING
Developer & Testing Forum

TF: Improvements of Metadata and other TF modules

By

Matvey Kraposhin
(kraposhin.online)

https://lfnetworking.org

The talk overview

- an implementation of IPv6 version of TF Metadata service
  - what is Metadata service?
  - how data passes through the TF data plane?
  - how the current implementation works?
  - new implementation key points
  - how to use new code?
- some considerations about improvement of VxLAN capabilities
  - a few words about TF data structures for storing routing information
  - some notes on the TF control plane main entities
  - the current implementation of VxLAN in TF
  - limitations of the current implementation
  - ideas for the improved implementation of the VxLAN component
- concluding remarks

LF NETWORKING

LFN Developer & Testing Forum

an implementation of IPv6 version of TF Metadata service

Objectives

Enable access to the Metadata service in Tungsten Fabric via IPv6 protocol

Nowadays it is possible only via IPv4 (169.254.169.254)

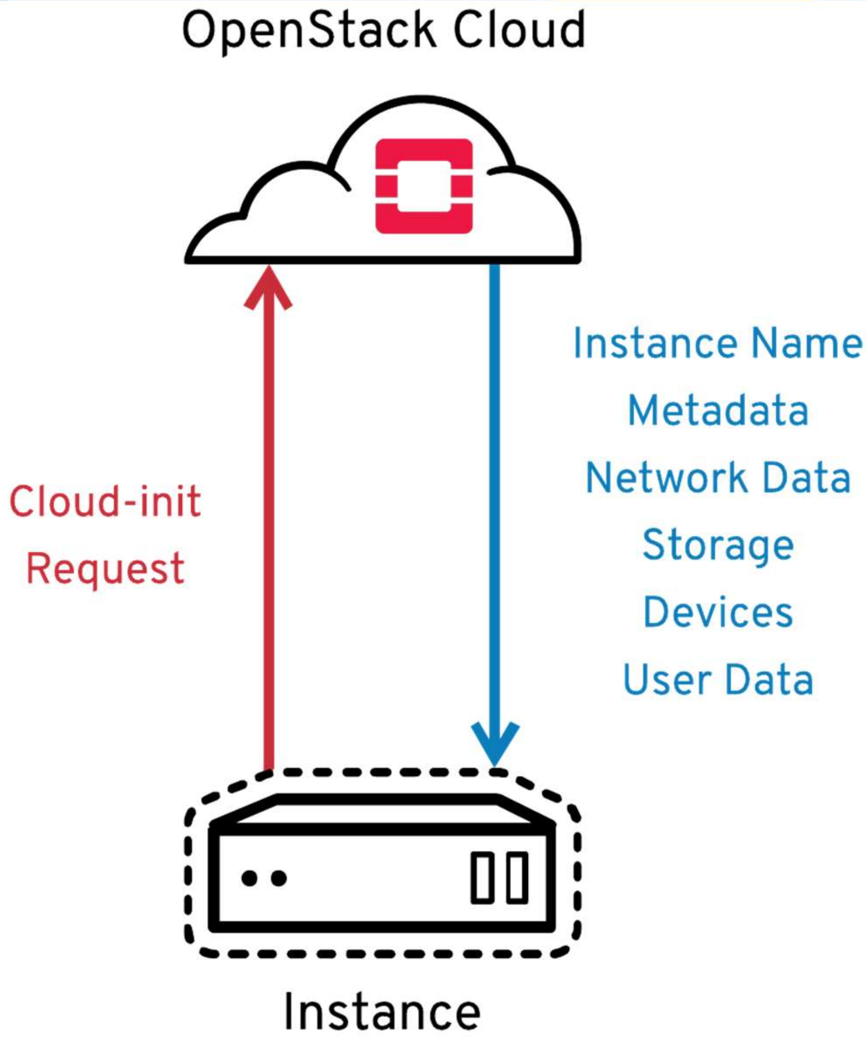
Motivation

1. OpenStack Metadata service provides a mean for instances to retrieve instance-specific data via the REST API. It can use both with IPv4 and IPv6 stacks:

- 169.254.169.254;
- fe80::a9fe:a9fe.

2. Most of other TF' link local services can use both IPv4 and IPv6

3. IPv6 is a "must" technology today in the SDN world



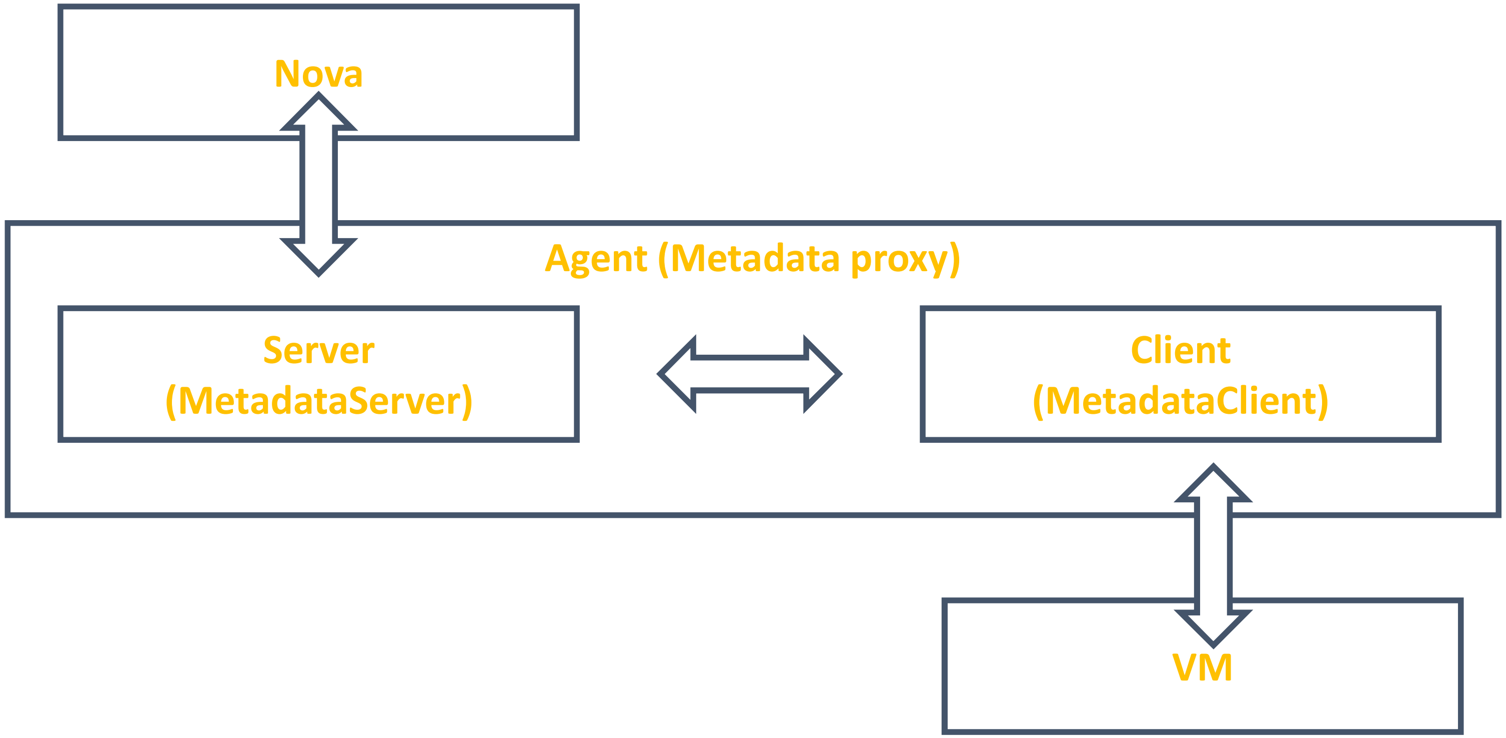
Metadata

LF NETWORKING
LFN Developer & Testing Forum

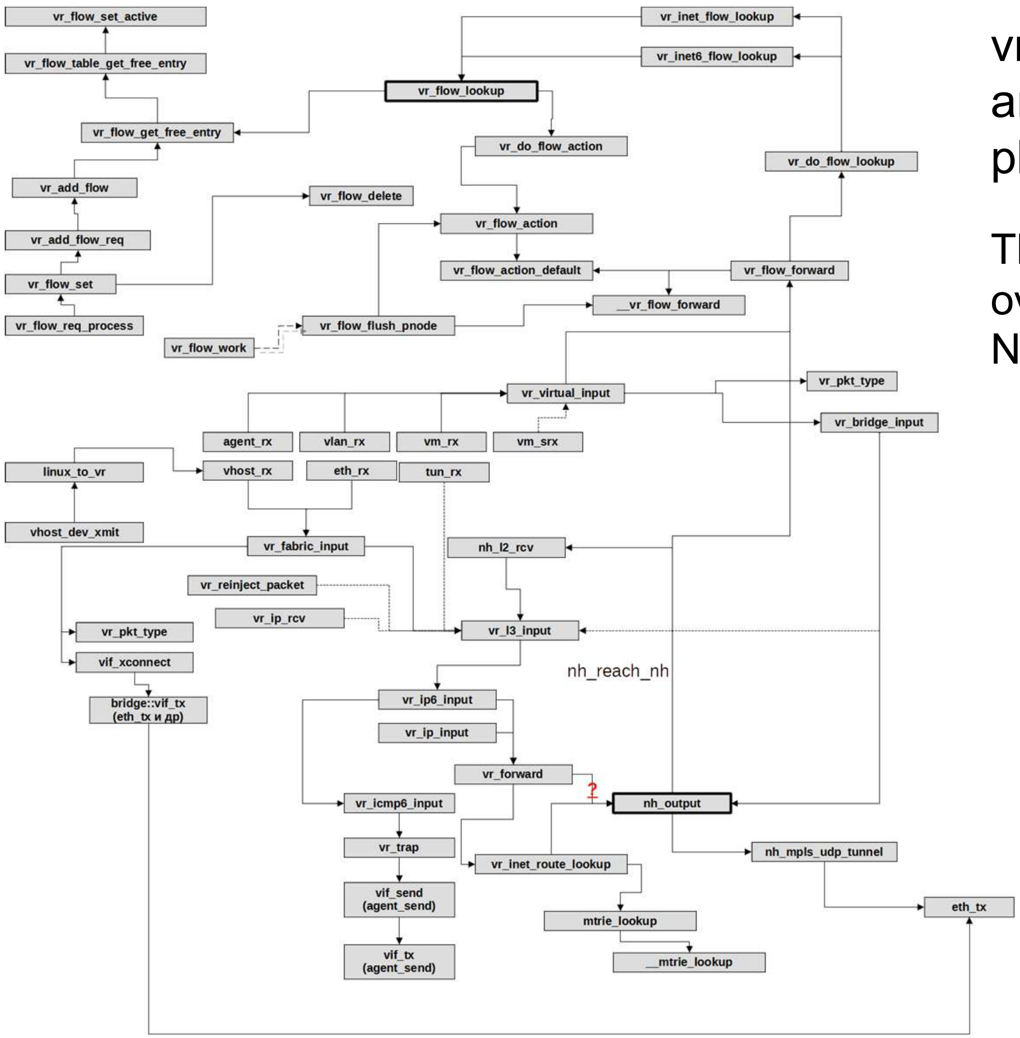
- Metadata is a tool to inject data into VM
- Types of Metadata :
  - Meta-Data
  - User-Data
  - Vendor-Data
  - Network-Data
- OpenStack Metadata FAQ
- TF docs
- All links can be found in the TF Metadata6 spec: https://github.com/tungstenfabric/tf-specs/blob/master/An_IPv6_Metadata_proxy_for_the_TF.md

TF metadata service

Tungsten Fabric metadata service actually is essentially a proxy server. It relays requests and responses from a VM to the Nova.



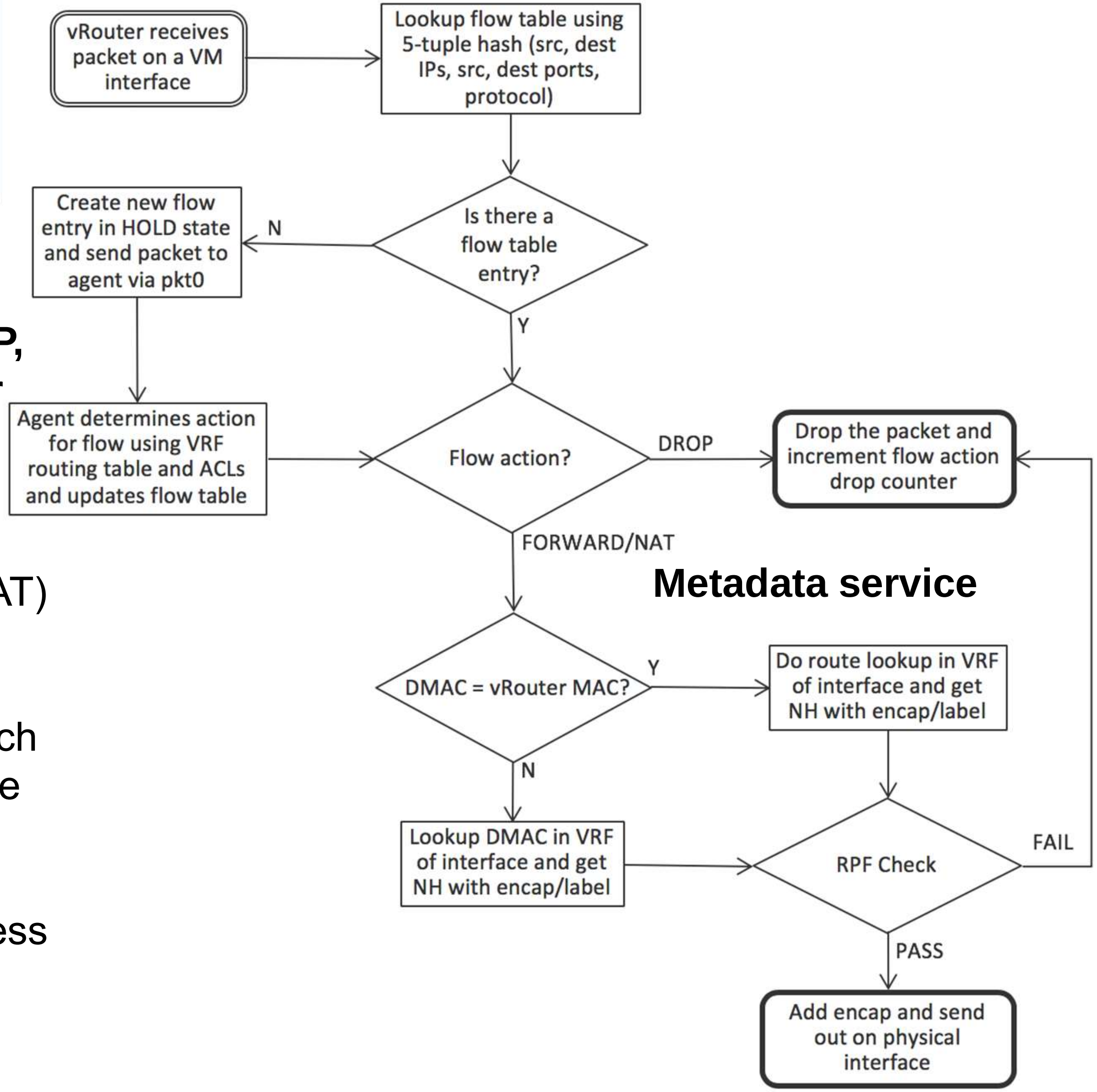
Nuances of requests relaying



vrouter can be considered as a large pinball desk with an IP packet instead of metallic ball and TF control-plane as levers

The most intricate part here is forwarding of messages from overlay network of VMs to underlay network where the Nova resides

NAT for Metadata



DNS, DHCP, ICMP, and other similar services

Metadata service

Metadata service uses NAT (source, destination and PAT) to forward messages to overlay network.

NAT is also used to identify VM in the Nova service: each VM is assigned in the overlay a unique IP address in the range 169.254.0.0/16.

When message comes to / from this address, the address is converted to credentials (UUID).

IPv6 NAT is not implemented in Tungsten Fabric

TF Link Local Services

DNS **DNS6**	General idea. Intercept an incoming message, generate a reply to it and send the reply back. The incoming message is intercepted at the packet analysis stage when the new flow is created by a request from vrouter
DHCP **DHCP6**	
ICMP **ICMP6**	

Work at L3/L2 levels

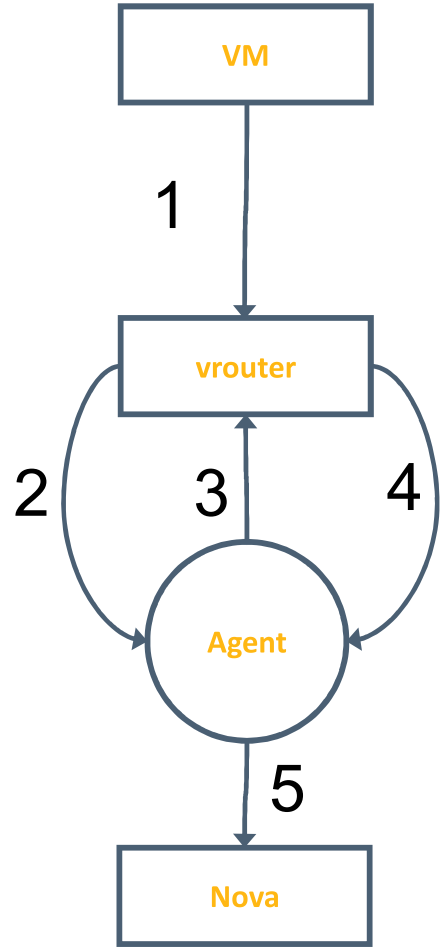
Metadata General idea.
Make a proxy: create 2 communicating TCP connections (one for interaction with the actual client, second for interaction with Nova)

To connect over- and underlay networks, use networks port and network d/s address translation

Work at L4/L3 levels

Current version of Metadata

1. VM sends request to the metadata server (169.254.169.254)
2. vrouter requests for flow between VM and 169.254.169.254
3. Parameters of NAT/PAT are sent to vrouter by the agent
4. vrouter sends message to Agent using NAT/PAT. UUID of a VM is obtained via LL (169.254.0.0/16) address.
5. Agent sends request to Nova

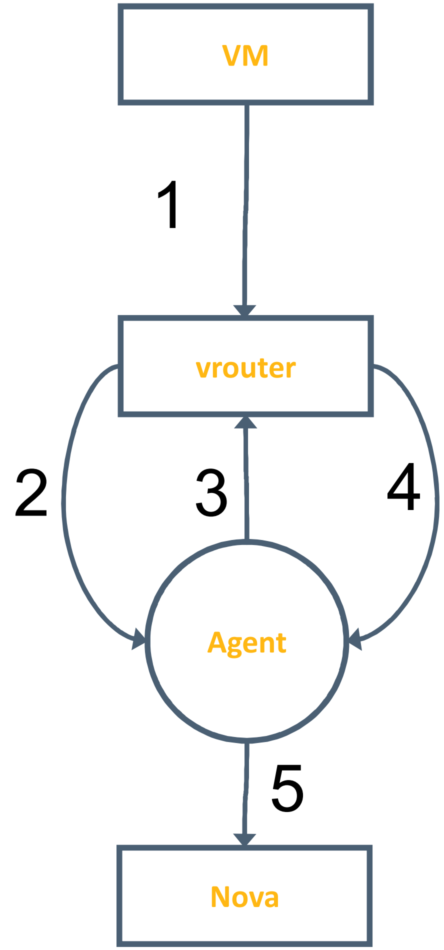


Modified version of Metadata

LF NETWORKING
LFN Developer & Testing Forum

1. VM sends request to the metadata6 server (fe80::a9fe:a9fe)
2. vrouter requests for flow between VM and fe80::a9fe:a9fe
3. New routes + 1 record linking VM LL IPv6 and VM UUID are created by the Agent
4. vrouter sends message to Agent using new routes
5. Agent sends request to Nova (using VM UUID identified at step 3).

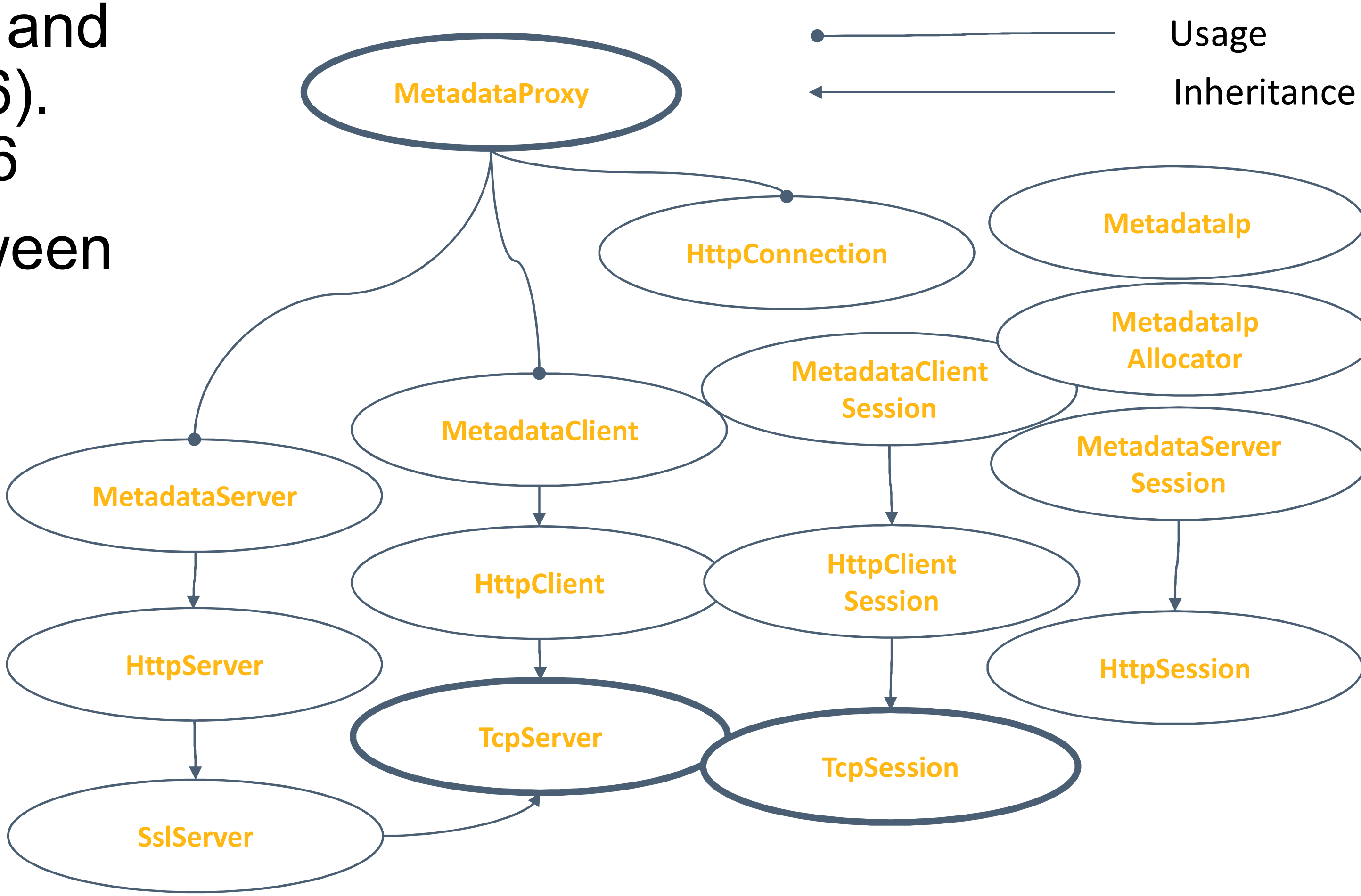
Key idea: use package analysis to detect metadata requests (like in DNS, DHPC, etc) and TCP proxyingto exchange data between user and NOVA



Challenges

1. Modification at both L4 and L3 levels, including NAT(6). TF still doesn't have NAT6

2. Complex relations between classes used to redirect requests to NOVA and to retrieve reply back



Demanded code changes

LFN Developer & Testing Forum

The changes embraced next parts of vrouter and Agent:

- MetadataProxy (Http/Tcp servers for IPv4 and IPv6)
- InterfaceTable (find creds using ll ipv6 address)
- PktFlowInfo (packet interception, routes announcement)
- TcpServer, TcpSession (support for IPv6)
- address_util.cc: support for IPv6 in ResolveCanonicalNameIPv6(…)
- vrouter: packet IPv6 forwarding between underlay and overlay was enabled (just 1 line of code)

```diff
 bool
 vr_is_ipv6_underlay_enabled(void)
 {
-    return vr_ipv6_underlay_enabled;
+    return vr_ipv6_underlay_enabled ||
+        vr_force_ipv6_underlay_enabled;
 }
```

Source code to intercept incoming metadata request

```
void PktFlowInfo::IngressProcess(const PktInfo *pkt, PktControlInfo *in,
                                 PktControlInfo *out) {
```

```
" << std::endl;
MetadataProxy *metadata_proxy = agent->services()->metadataproxy();
if(metadata_proxy && pkt->ip_saddr.is_v6()
   && pkt->ip_daddr.to_string() == metadata_proxy->Ipv6ServiceAddress().to_string()) {
   Ip6Address ll_ip = pkt->ip_saddr.to_v6();
   std::cout<< "A request to fe80::a9fe:a9fe"
            << " from " << ll_ip.to_string()
            << " with vrf " << (in->vrf_ ? in->vrf_->GetName() : "NONE") << std::endl;
    //Step 1. Check port
    uint16_t nova_port, linklocal_port;
    Ip4Address nova_server, linklocal_server;
    std::string nova_hostname;
    if (agent->oper_db()->global_vrouter()->FindLinkLocalService(
        GlobalVrouter::kMetadataService, &linklocal_server, &linklocal_port,
        &nova_hostname, &nova_server, &nova_port))
    {
        std::cout << "Reseting port to: " << linklocal_port << std::endl;
        metadata_proxy->ResetIp6Server(linklocal_port);
    }
    //Step 2.
    metadata_proxy->AnnounceMetaDataLinkLocalRoutes(vm_port,
        ll_ip, in->vrf_);
}
```

Steps to enable Metadata6

- Enable forwarding of IPv6 packets between underlay and overlay:
- Connect a VMI with the IPv6 address to a VM
- Start using Metadata6

tf-controller commit:
https://github.com/tungstenfabric/tf-controller/commit/61c062d0a8e51b1826002d4d7bd0ce33da1cf986

tf-common commit:
https://github.com/tungstenfabric/tf-common/commit/c0527f40624854c610d1f7a3bfd4d9d515693e23

tf-vrouter commit:
https://github.com/tungstenfabric/tf-vrouter/commit/97cbacb1f12151efb65717adf5dd73aae21465e4

tf-specs commit:
https://github.com/tungstenfabric/tf-specs/commit/d0fdc712db8ccc755ad392464c5af835f1ec927e

Case when we have only IPv6 VMIs

Enable IPv6 between underlay and overlay (on the compute node):

```
# echo "Y" > /sys/module/vrouter/parameters/vr_force_ipv6_underlay_enabled
```

Prepare virtual-network with IPv6 subnet, create new IPv6 instance and link it with VMI

K-net	ee80::/10	-

Subnets

Allocation Mode	User Defined	
Subnet(s)	CIDR	Gateway
	ee80::/10	ee80::1

Details

vmi1 (8a66c114-59ce-4163-bae7-2355dc6c9362)	-

Port Detail

Network	K-net
UUID	8a66c114-59ce-4163-bae7-2355dc6c9362
Name	vmi1
Display Name	vmi1
Admin State	Up
MAC Address	02:8a:66:c1:14:59
Fixed IPs	ee80::11
Security Groups	Enabled default

Start a virtual machine with connected IPv6 VMI

```
[fedora@vm-test ~]$ ssh ee80::11 -i ~/.ssh/mypair
The authenticity of host 'ee80::11 (ee80::11)' can't be established.
ED25519 key fingerprint is SHA256:wecwhed4P+28uoYj0q3pvIBlvuTr0Gq62Wr+QRrIYcM.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ee80::11' (ED25519) to the list of known hosts.
Last login: Mon Jun  5 20:25:23 2023 from ee80::5
[fedora@vm-test6-1 ~]$ ping fe80::a9fe:a9fe%eth1
PING fe80::a9fe:a9fe%eth1(fe80::a9fe:a9fe%eth1) 56 data bytes
64 bytes from fe80::a9fe:a9fe%eth1: icmp_seq=1 ttl=64 time=1.10 ms
64 bytes from fe80::a9fe:a9fe%eth1: icmp_seq=2 ttl=64 time=0.235 ms
64 bytes from fe80::a9fe:a9fe%eth1: icmp_seq=3 ttl=64 time=0.246 ms
64 bytes from fe80::a9fe:a9fe%eth1: icmp_seq=4 ttl=64 time=0.242 ms
^C
--- fe80::a9fe:a9fe%eth1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3076ms
rtt min/avg/max/mdev = 0.235/0.455/1.097/0.370 ms
[fedora@vm-test6-1 ~]$
```

```
[fedora@vm-test6-1 ~]$ curl  http://[fe80::a9fe:a9fe%eth1]:80
1.0
2007-01-19
2007-03-01
2007-08-29
```

Diagnostics

LF NETWORKING
LFN Developer & Testing Forum

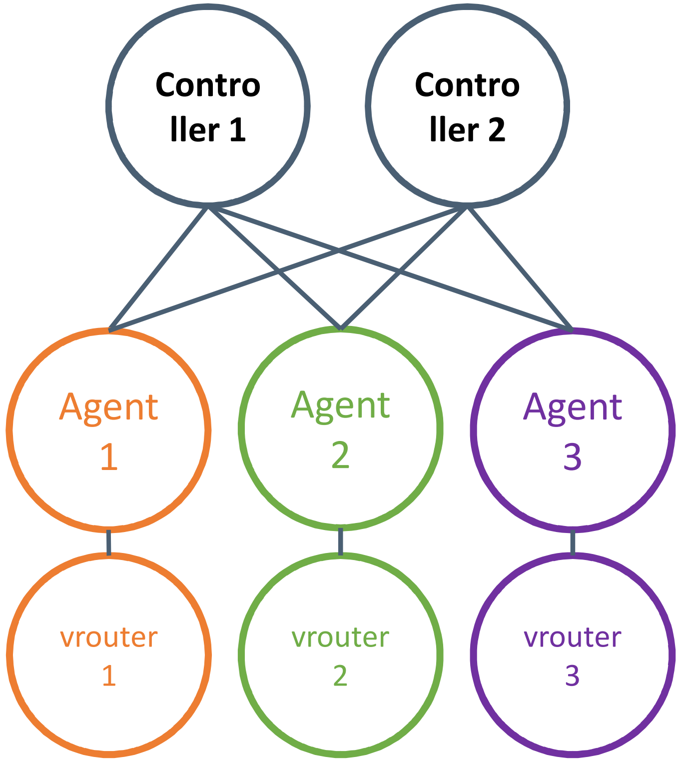
- Check that the IPv6 link-local address of the Metadata proxy server (fe80::a9fe:a9fe) has appeared in the corresponding VRF table.
- Security groups
- Connect VM to VMI, ping -6 fe80::a9fe:a9fe%eth1
- Check appearance of corresponding routes (ll ipv6 of eth1) in the VMI VRF table and in the fabric vrf (__default__)
- Check that corresponding vrouter option is turned on (enable /sys/module/vrouter/parameters/vr_force_ipv6_underlay_enabled)
- tcpdump (vhost0, eth1, tap-…)

LFN Developer & Testing Forum

some considerations about improvement of VxLAN capabilities

TF Routing information

- Tungsten Fabric as a highly specialized tables processor:
  - The Controller table (centralized but with duplicates)
  - The Agent table (part of controller table)
  - The vrouter table (forwarding table)



Each Agent is responsible for storing of the part of the Controller's table. That part corresponds to local VM interfaces. The remaining part is stored as tunnels. Each controller table has as many copies as many controllers are present in the TF.

The TF agent table(s)

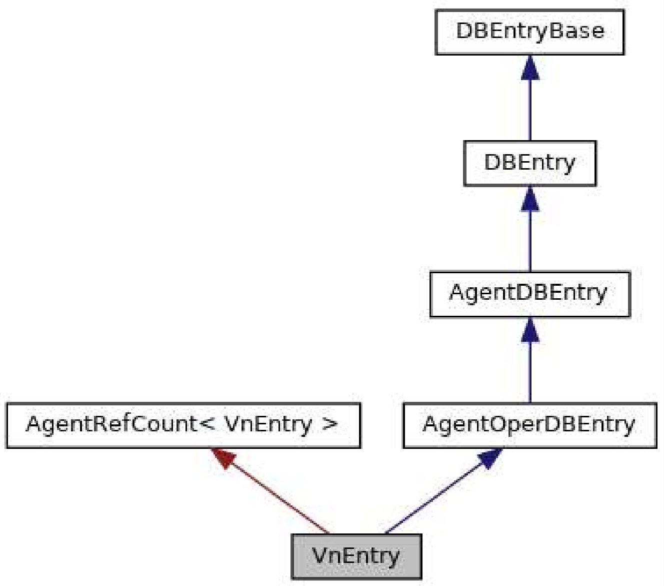
- Virtual Network (VN)
- Virtual Routing & Forwarding (VRF)
- Route tables (EVPN & INET)
- Route
- Prefix:
  - Prefix address – L3 (IP) and L2 (MAC)
  - Prefix length
- Path
- Nexthop
- Peer

Virtual Network

Virtual Network contains everything what is needed to define a connectivity between virtual machines (nodes):

- Access control list
- List of IPAMs
- Security logging objects
- Routing information (VRF table)

Each **Virtual Network** is stored in an object of the **VnEntry** class.
VnEntry objects are stored in **VnTable** class

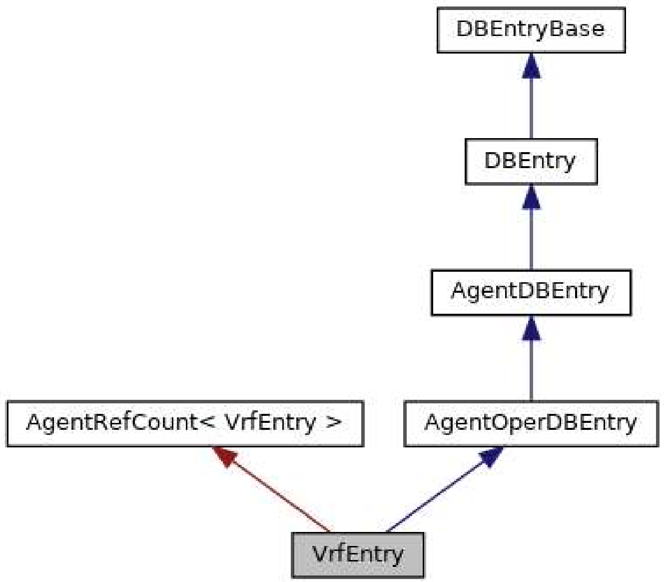


Virtual Routing & Forwarding

Virtual Routing & Forwarding instance defines tables to organize L2 and L3 reachability between nodes:

- Inet IPv4 table (L3)
- Inet IPv6 table (L3)
- EVPN table, which supports Type 2 and Type 5 routes for IPv4 and IPv6 protocols.
- Other tables

Each **VRF** instance is stored in an object of the **VrfEntry** class. **VrfEntry** objects are stored in **VrfTable** class

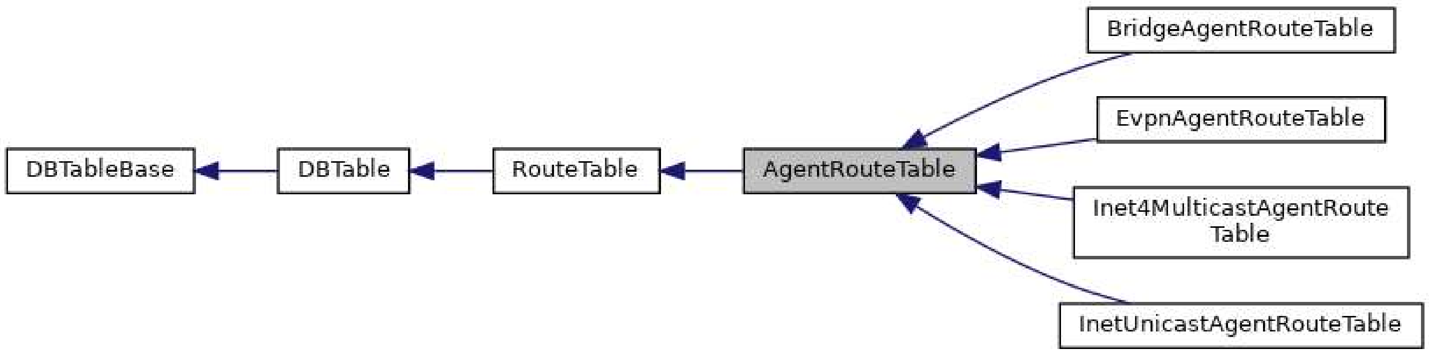


Route tables

Route table defines list of records to reach nodes by their IP or MAC address:

- a list of routes
- tools to find records

Each **Route table** instance is stored in an object of the **AgentRouteTable** class. **AgentRouteTable** objects are stored in **VrfEntry** class.

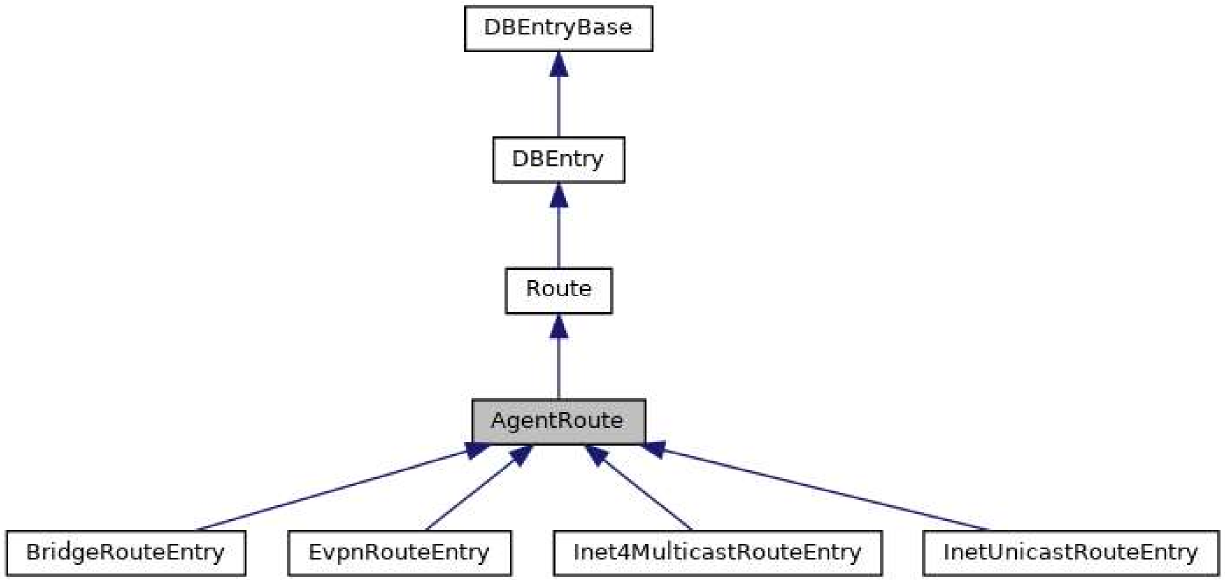


Route

Route is a record that is characterized by:

- a destination IP prefix (in case of L3)
- a destination MAC (in case of L2)
- list of paths to a destination
- tools to process properties of a route

Each **Route table** instance is stored in an object of the **AgentRoute** class. **AgentRoute** objects are stored in **AgentRouteTable** class.



Prefix

Prefix is a sequence of bytes defining address of a network node:

- it is 6-byte in case of MAC address
- it is 4-byte in case of IPv4 address + prefix length
- it is 16-byte in case of IPv6 address + prefix length

Prefix is a synonym for address.

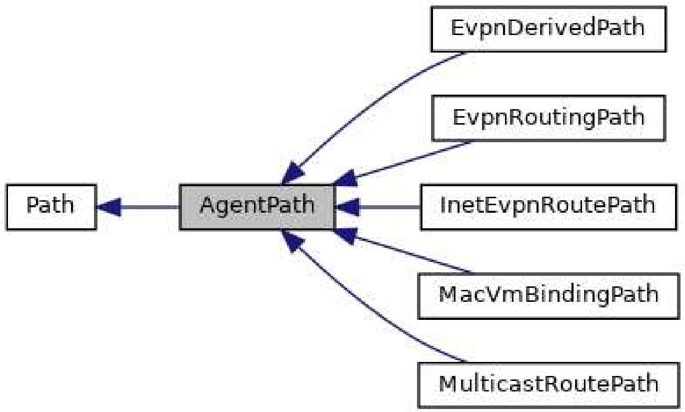
MAC addresses are stored in **MacAddress** class. IP addresses are stored in **IpAddress** class. The latter one is an alias for **boot::ip::address**

Path

Path contains information to reach a prefix within the route and tools to manipulate it:

- Nexthop
- Peer
- Tags list, security groups list

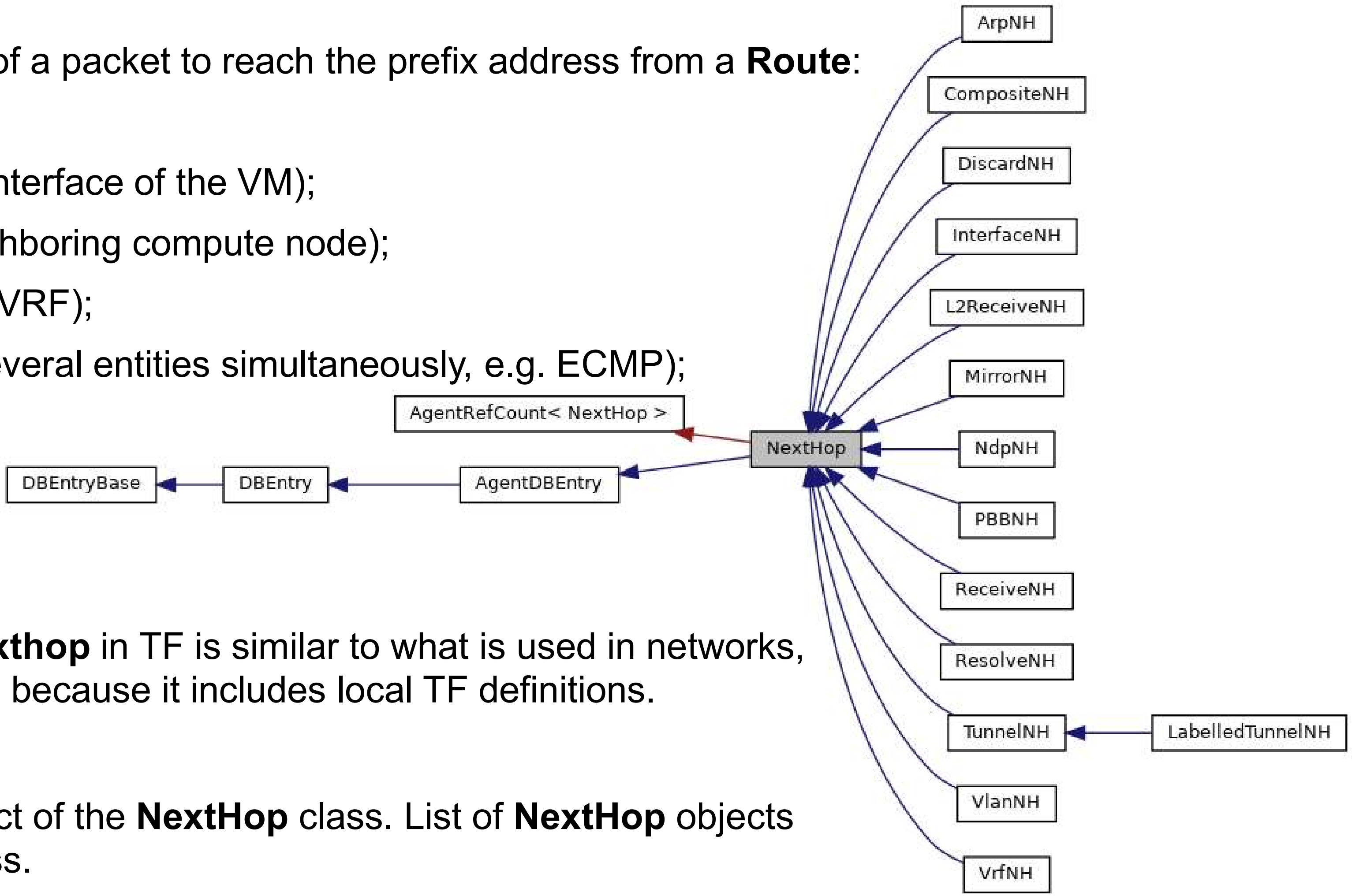
Each **Path** instance is stored in an object of the **AgentPath** class. List of **AgentPath** objects is stored in the **AgentRoute** class.



Nexthop

Nexthop defines next destination of a packet to reach the prefix address from a **Route**:

- type of Nexthop:
  - the Interface (points to an interface of the VM);
  - the Tunnel (points to a neighboring compute node);
  - the VRF (points to another VRF);
  - the Composite (points to several entities simultaneously, e.g. ECMP);
  - other types;
- type-specific information
- manipulation tools



While the general sense of the **Nexthop** in TF is similar to what is used in networks, it is slightly different from the latter, because it includes local TF definitions.

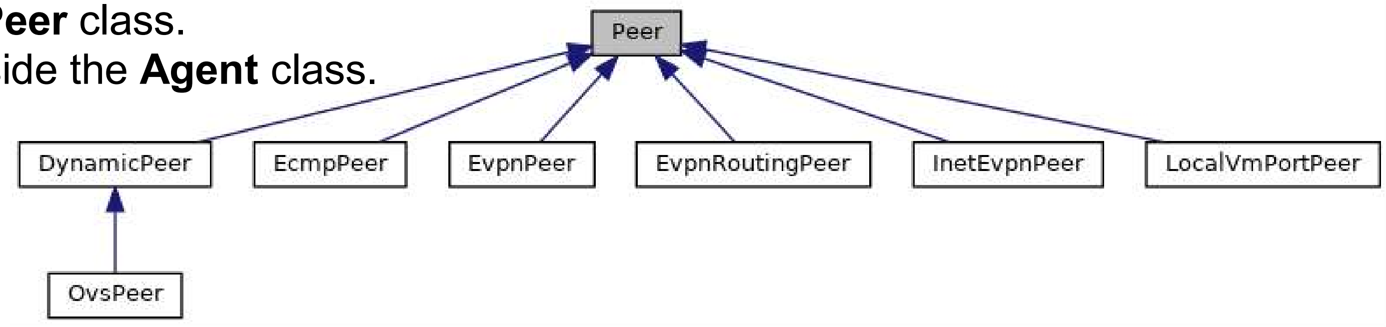
Each **Nexthop** is stored in an object of the **NextHop** class. List of **NextHop** objects is stored in the **NextHopTable** class.

Peer

Peer shows origin where a path originates from or purpose of a path. Peers can be of next types:

- LOCAL_VM_PEER characterises paths ending in VMs connected to a TF compute (e.g., interfaces);
- BGP_PEER characterises paths which are announced via BGP/XMPP protocol (e.g., tunnels);
- EVPN_ROUTING_PEER characterises paths which are used for routing between networks;
- ECMP_PEER characterises paths pointing to several VMs connected to a single TF compute (e.g. several interfaces);
- and others.

Each **Peer** is stored in an object of the **Peer** class.
Some Peers are allocated as objects inside the **Agent** class.



An example of a route table

```
Evpn Type 5 table:
  IP:10.0.0.250, path count = 1, ethernet_tag = 0
    NH: Tunnel to 172.16.0.29 rewrite mac 00:50:56:01:af:7f, Peer:172.16.0.22
  IP:10.1.1.0, path count = 2, ethernet_tag = 0
    NH: Tunnel to 172.16.0.26 rewrite mac 00:00:00:00:00:00, Peer:172.16.0.22
    NH: InterfaceNH : tap09041492-09, Peer:LocalVmExportPeer
  IP:10.1.1.3, path count = 2, ethernet_tag = 0
    NH: Composite NH, Peer:172.16.0.22
    n components=2
    NH: InterfaceNH : tap6dbb2828-a9, Peer:LocalVmExportPeer
  IP:10.1.1.10, path count = 1, ethernet_tag = 0
    NH: Tunnel to 172.16.0.29 rewrite mac 00:50:56:01:af:7f, Peer:172.16.0.22
  IP:10.1.1.31, path count = 2, ethernet_tag = 0
    NH: InterfaceNH : tap09041492-09, Peer:172.16.0.22
    NH: InterfaceNH : tap09041492-09, Peer:LocalVmExportPeer
  IP:10.1.1.41, path count = 2, ethernet_tag = 0
    NH: InterfaceNH : tap38253589-5b, Peer:172.16.0.22
    NH: InterfaceNH : tap38253589-5b, Peer:LocalVmExportPeer
  IP:10.1.1.201, path count = 2, ethernet_tag = 0
    NH: InterfaceNH : tap09041492-09, Peer:172.16.0.22
    NH: Composite NH, Peer:LocalVmExportPeer
    n components=2
  IP:10.10.10.0, path count = 1, ethernet_tag = 0
    NH: Tunnel to 172.16.0.28 rewrite mac 00:00:00:00:00:00, Peer:172.16.0.22
  IP:10.100.100.0, path count = 2, ethernet_tag = 0
    NH: Tunnel to 172.16.0.26 rewrite mac 00:00:00:00:00:00, Peer:172.16.0.22
    NH: InterfaceNH : tap09041492-09, Peer:LocalVmExportPeer
```

- A **peer** together with a **nexthop** make up the **path**.
- List of **path**s make up a **route**. Combination of a **peer** type and a nexthop is unique within a **route**.
- **Route**s make up a **route table**.
- **Route tables** make up a **VRF** instance.

Routes leaking

Routes leaking is a procedure of routes synchronization between two tables according to some predefined rules.

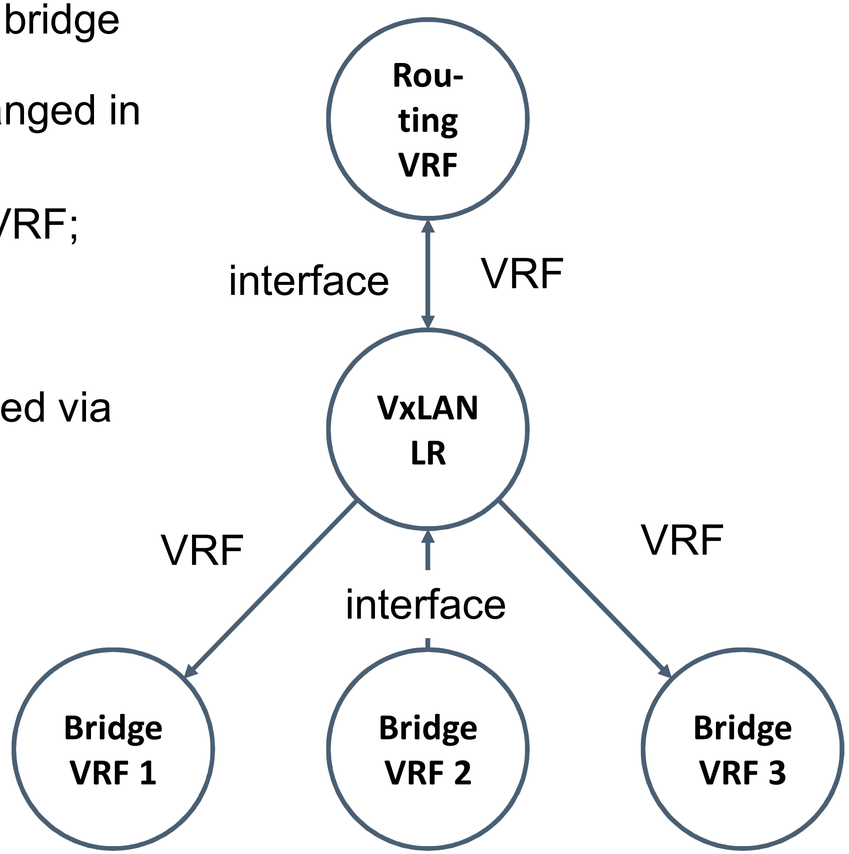
Routes can be synchronized between tables in one VRF instance or between VRF instances.

Table1		Table2
Prefix1	→	Prefix1
Path1: Nexthop1 Peer1 Path2: Nexthop2 Peer2		Path1: Nexthop1 Peer1 Path2: Nexthop2 Peer2
Prefix1	→	Prefix1
Path3: Nexthop3 Peer1 Path4: Nexthop4 Peer2		Path3: Nexthop3 Peer1 Path4: Nexthop4 Peer2

VxLAN routes leaking in TF

VxLAN implementation of TF is a routes leaking mechanism between the routing VRF instance connected to a VxLAN logical router (LR) and bridge VRF instances connected to this LR.
Within each routing – bridge pair of VRF instances routes are exchanged in accordance with these rules:

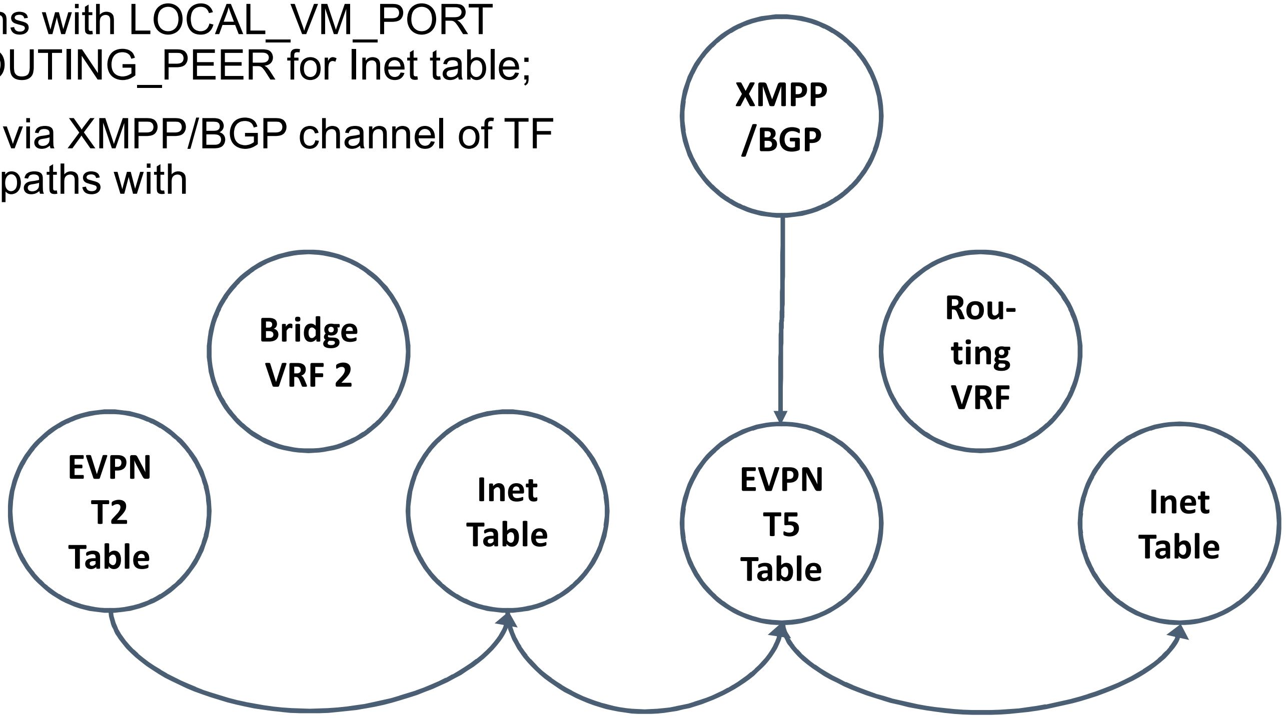
- bridge VRFs routes with interface path are copied in the routing VRF;
- advertisement of an interface route in the routing VRF yields advertisement of a VRF NH path in other bridge VRF instances;
- tunnel paths (path with tunnel nexthop and BGP_PEER) are copied via XMPP/BGP channel of TF.



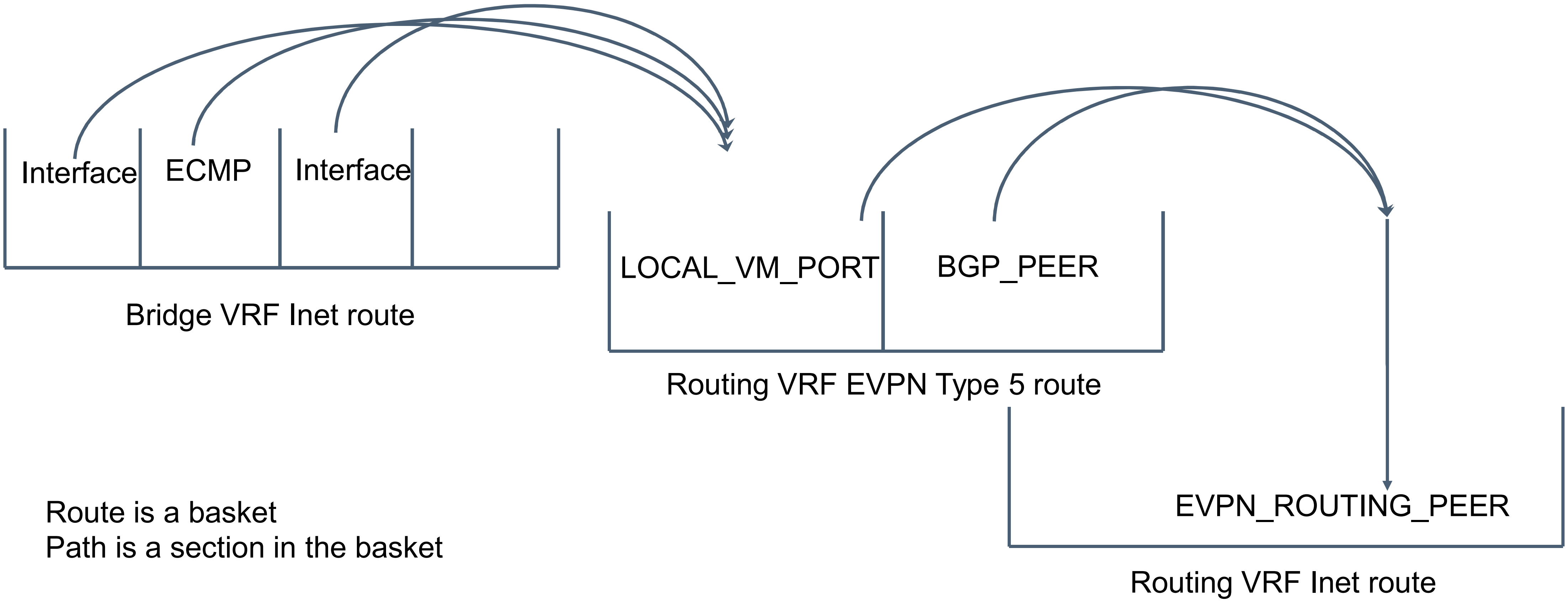
Implementation of VxLAN in TF

VxLAN implementation.

- a change in a bridge EVPN Type 2 tables creates VRF nexthop in the Inet table of the bridge VRF;
- Interface (LOCAL_VM_PORT/ECMP_PEER peer) paths are copied from the bridge VRF to the routing VRF into paths with LOCAL_VM_PORT peer for EVPN Type 5 table and EVPN_ROUTING_PEER for Inet table;
- tunnel paths (with BGP_PEER) are copied via XMPP/BGP channel of TF to the routing VRF EVPN Type 5 table into paths with LOCAL_VM_PORT.



An analogy with baskets



Route is a basket
Path is a section in the basket

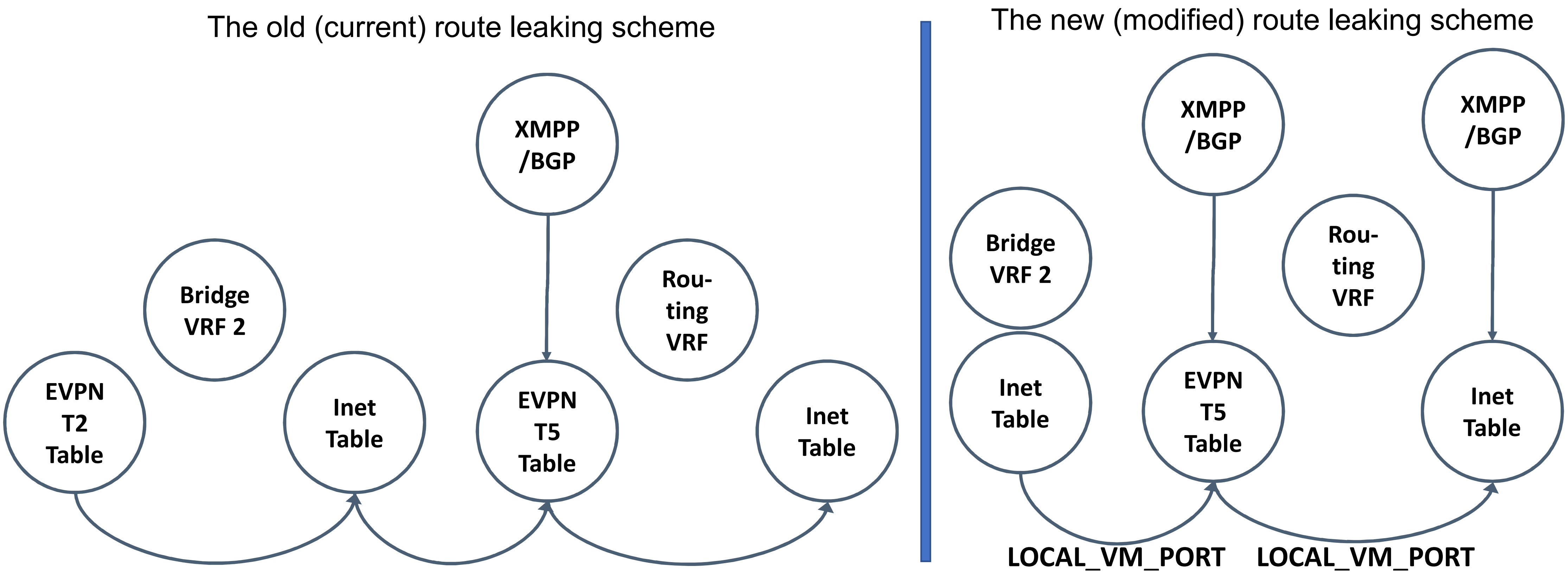
Limitations of the current approach

1. Since route leaking is triggered by changes in EVPN Type 2 and Inet tables, then only L2-L3 routes could leak.
2. Routes originating from special types of IP instance cannot leak: Floating IP, L3 Allowed Address Pairs, BGP-as-a-Service, etc.
3. Deletion of one path in a bridge VRF route might lead to deletion of the whole corresponding route in the routing VRF (Since BGP_PEER and LOCAL_VM_PORT are mixed EVPN_ROUTING_PEER in the Inet table of the routing VRF instance).
4. Synchronization problems: if, for example, a tunnel path arrives later than interface one, it rewrites interface, etc
5. Composite (ECMP) routes are not fully supported.

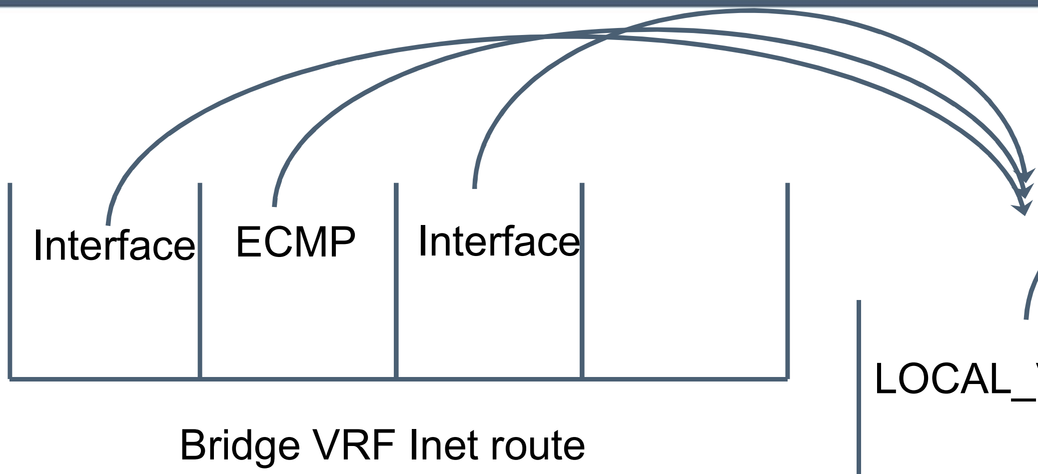
Experiment.
Using TF of all versions except **master branch**, create several VMI, link them to 1 IP instance and connect VMIs to virtual machines on different compute nodes. In this case, ECMP routes in bridge VRF instances and the routing VRF instance will be different.
Issue No. 4 was partially resolved in the **master branch**.

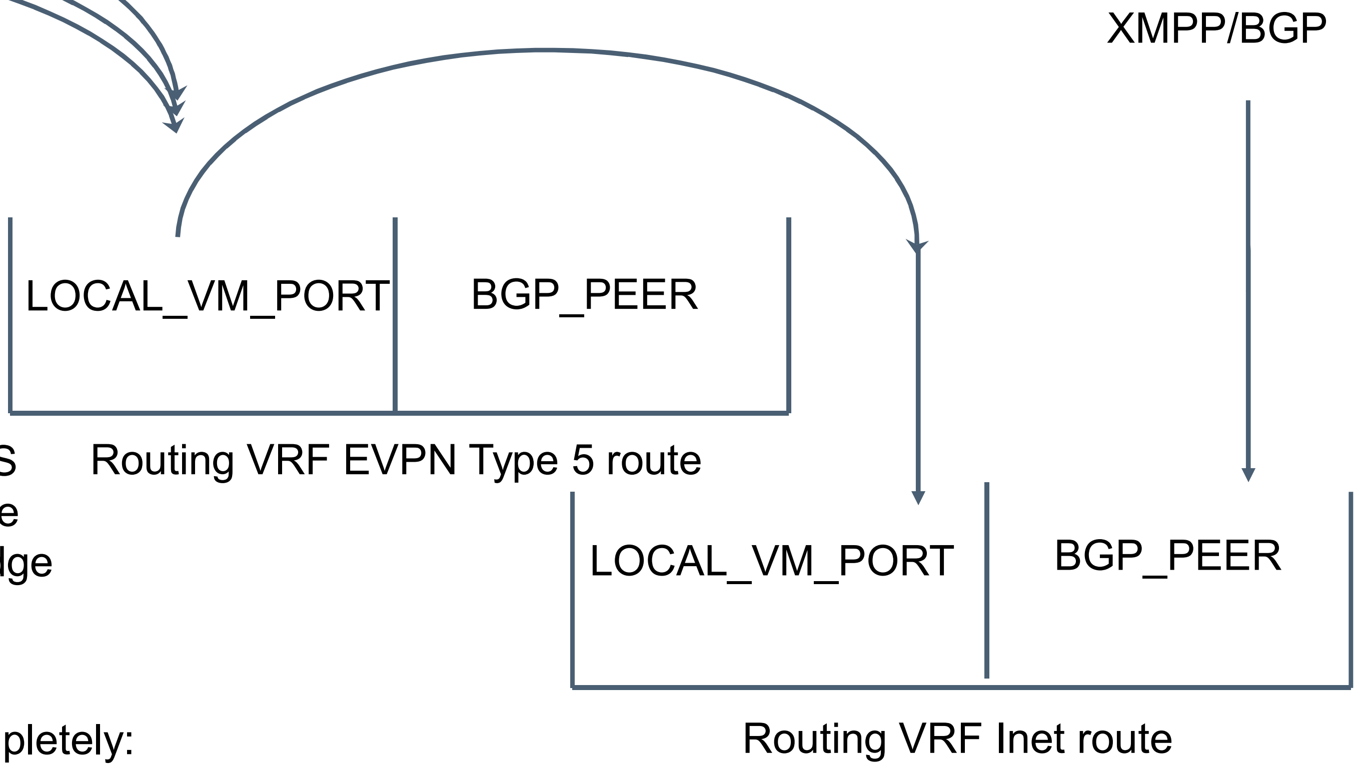
Improved route leaking scheme

1. Trigger route leaking only by changes in Inet bridge VRF tables.
2. Do not leak BGP_PEER routes between EVPN Type5 and Inet tables of the routing VRF.



Analogy with baskets for the modified case





In this case FIP/AAP leak naturally, BGPaaS routes leaking requires minor changes to the code. ECMP routes are now identical in bridge and routing VRF instances.

2 parts were of TF have been rewritten completely:

1) Leaking of LOCAL_VM_PORT routes
2) Input of routes from XMPP/BGP

Implementation state

LFN Developer & Testing Forum

1. The new version of overall algorithm for routes leaking for VxLAN has been developed.
2. Preliminary version which allows FIP/AAP/BGPaaS routes leaking has been implemented.
3. Code refactoring is ongoing.
4. Next stage: unit tests and functional tests.

LFN NETWORKING
LFN Developer & Testing Forum

concluding remarks

Acknowledgements

- Yurii Konovalov for inspiring ideas and overall supervision.
- Andrey Pavlov for CI support and limitless patience.
- My colleagues for valuable conversations.
- Sayali Mane, Casey Cain, Nick Dave for organizational support.
- Juniper corporation for the TF technology.
- And all participants of this D&TF session.

Russian TF group

Concluding remarks

- Extension of TF’s Metadata service to IPv6 protocol allows injection of data into VM in pure IPv6 networks
- The approach can be re-used for custom TF link-local services that are still IPv4
- The new algorithm of route leaking for Tungsten Fabric VxLAN has been proposed
- This algorithm allows to support more ways of routes advertising, such as Floating IP, BGP-as-a-Service, static interface routes, etc

Russian TF group

Click to edit Master title style