networking-bagpipe Documentation

Release 19.0.1.dev3

OpenStack Foundation

Nov 29, 2024

CONTENTS

Driver and agent code to use BaGPipe lightweight implementation of BGP-based VPNs as a backend for Neutron.

- Free software: Apache license
- Documentation: https://docs.openstack.org/networking-bagpipe/latest/
- Source: http://opendev.org/openstack/networking-bagpipe
- Bugs: https://bugs.launchpad.net/networking-bagpipe
- Release notes: https://docs.openstack.org/releasenotes/networking-bagpipe/

CHAPTER ONE

OVERVIEW

BGP-based VPNs rely on extensions to the BGP routing protocol and dataplane isolation (e.g. MPLS-over-x, VXLAN) to create multi-site isolated virtual networks over a shared infrastructure, such as BGP/MPLS IPVPNs (RFC4364) and E-VPN (RFC7432). They have been heavily used in IP/MPLS WAN backbones since the early 2000s.

These BGP VPNs are relevant in the context of Neutron, for two distinct use cases:

1. creating reachability between Neutron ports (typically VMs) and BGP VPNs outside the cloud datacenter (this use case can be relevantindependently of the backend chosen for Neutron)
2. leveraging these BGP VPNs in Neutrons backend, to benefit from the flexibility, robustness and scalability of the underlying technology (as do other existing backends such as OpenContrail, Nuage Networks, or Calico although the latter relies on plain, non-VPN, BGP)

BaGPipe proposal is to address these two use cases by implementing this protocol stack both the BGP routing protocol VPN extensions and the dataplane encapsulation in compute nodes or possibly ToR switches, and articulating it with Neutron thanks to drivers and plugins.

The networking-bagpipe package includes:

- for use case 1: backend code for Neutrons BGPVPN Interconnection service plugin (networking-bgpvpn) ; only compute node code (agent and BGP) is in networking-bagpipe, the Neutron server-side part, being currently in networking-bgpvpn package)
- for use case 2: a Neutron ML2 mechanism driver (base Neutron networks), a networking-sfc driver (service chaining)
- compute code common to both: agent extensions for Neutron agent (linuxbridge or openvswitch) to consolidate and pass information via its REST API to BaGPipe-BGP (a lightweight BGP VPN implementation)

CHAPTER TWO

USING BAGPIPE

2.1 Design overview

The common design choices underlying bagpipe architecture are:

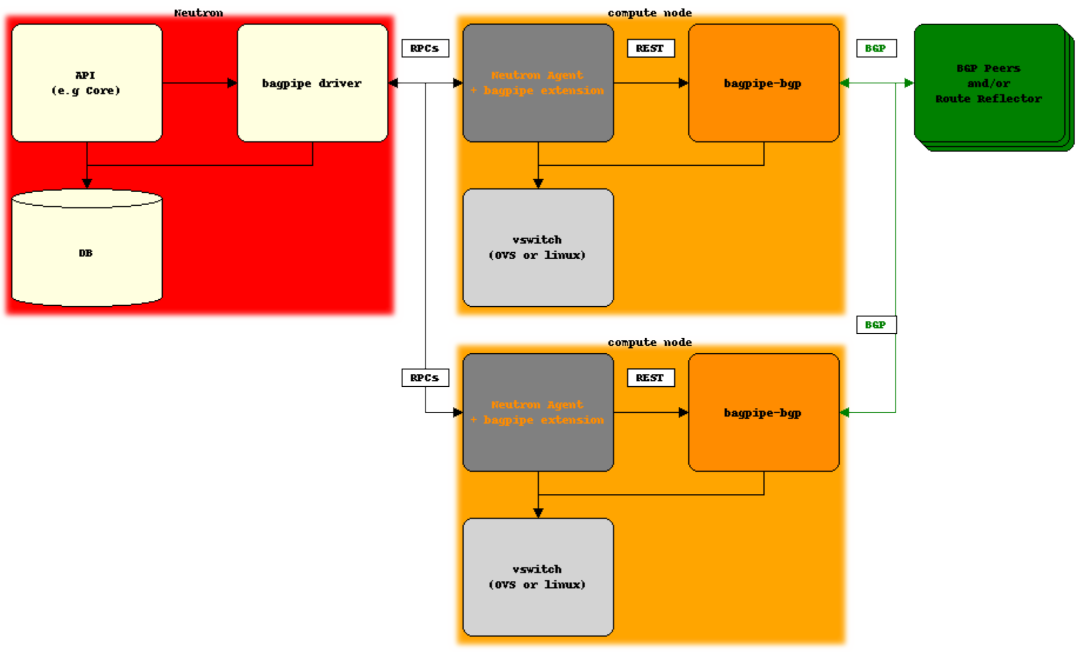
a. on Neutron server, allocate and associate BGP VPN constructs necessary to realize Neutron API abstractions: network, router, service chain, BGP VPN interconnection, etc.

b. pass the information about these BGP VPN constructs to the compute node agent via Openstack Neutron message bus (typically, but not necessarily RabbitMQ)

c. on compute node, a bagpipe extension of the Neutron agent (OVS or linuxbridge) passes the information to the local implementation of BGP VPN extensions (*BaGPipe-BGP*) that will advertise and receive BGP VPN routes and populate the dataplane accordingly

d. depending on the use cases, BGP VPN routes are exchanged between compute nodes, between compute nodes and DC gateway IP/MPLS routers, or both ; the strategy to scale this control plane will depend on the deployment context but will typically involve BGP Route Reflectors and the use of the RT Constraints pub/sub mechanism (RFC4684)

e. traffic is exchanged using an overlay encapsulation, with VXLAN as the typical choice for vswitch-to-vswitch, and MPLS-over-GRE or MPLS-over-UDP (future) as the target for vswitch-to-DC-gateway traffic



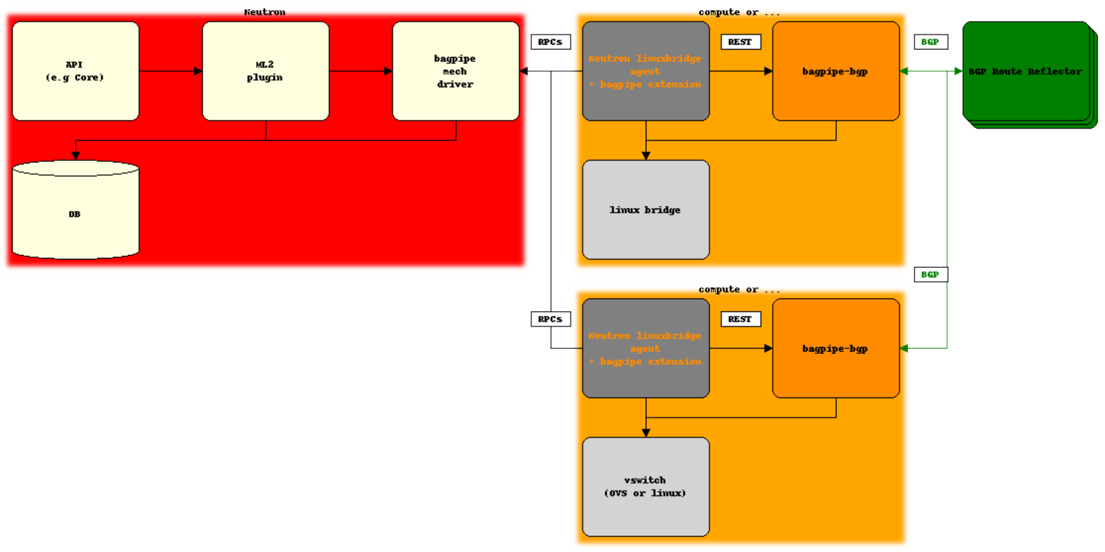
2.2 Applications

2.2.1 Neutron networks (ML2)

Note: This application is distinct from the use of BaGPipe to create IPVPN or E-VPN interconnections in the context of the BGPVPN Interconnection API (see below).

The `bagpipe` mechanism driver for Neutrons ML2 core plugin, when enabled along with the corresponding compute node agent extension, will result in Neutron tenant networks to be realized with E-VPN/VXLAN.

How it works is that a BGP VPN identifier (called a BGP Route Target) will be defined for each Neutron tenant network, derived from the VXLAN VNI a.k.a segmentation ID, and the `bagpipe` agent extension on compute nodes will setup a corresponding E-VPN instance with this identifier on the local *BaGPipe-BGP* instance on the compute node and attach VM ports to this instance as needed.



This solution is currently supported with the linux networking stack (i.e. with the `linuxbridge` agent enabled with bagpipe extension, and *BaGPipe-BGP* driver for the linux bridge VXLAN implementation). The approach would be easily extended to support OpenVSwitch as well.

Another way to understand this approach for someone coming with a Neutron ML2 background is that it is similar to the l2population mechanism except that the bridge forwarding entries are populated based on BGP VPN routes rather than based on information distributed in RPCs. This similarity comes with a difference: while l2population announces the information on one messaging topic, each compute node receiving information about all Neutron networks even the ones not present on its vswitch, the behavior with BaGPipe ML2 is that a compute node will only receive the mappings that it needs.

2.2.2 Neutron BGPVPN Interconnection

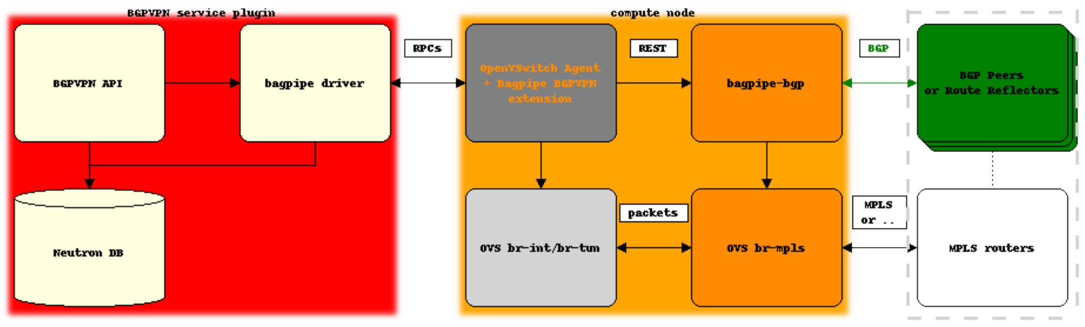
Note: This application is distinct from the use of BaGPipe to realize Neutron networks with BGP E-VPNs. `bagpipe` driver for networking-bgpvpn supports both IPVPNs and E-VPNs, but does not rely on `bagpipe` ML2 mechanism driver to do so.

In this application, `networking-bagpipe` aims at proposing a lightweight implementation of the BGPVPN Interconnection service, designed to work with the ML2 `openvswitch` or `linuxbridge` mechanism drivers (or as an alternative with the `bagpipe` ML2 mechanism driver).

When used along with the `openvswitch` or `linuxbridge` ML2 mechanism driver, it involves the use of:

- `bagpipe` driver for the BGPVPN service plugin (in networking-bgpvpn package)
- `bagpipe_bgpvpn` extension for the Neutron compute node agent (in this package)
- *BaGPipe-BGP* lightweight BGP VPN implementation (in this package)

Example with OVS agent:



2.2.3 Service Chaining (SFC)

For this application, `networking-bagpipe` provides a `bagpipe` driver for the `networking-sfc` that will result in service chains defined via the networking-sfc API, being realized with BGP VPN stiching, BGP VPN route redistribution and BGP Flowspec routes.

The components involved are:

- `bagpipe` driver for the `networking-sfc` service plugin (in this package)
- `bagpipe_sfc` extension for the Neutron compute node agent (in this package)
- *BaGPipe-BGP* lightweight BGP VPN implementation (in this package)

Note: This driver is still quite experimental, and still currently relies on using the `linuxbrige` agent along with the OVS dataplane driver for IPVPN in `bagpipe-bgp`.

2.2.4 Work in progress and future applications

Work in progress:

- BaGPipe ML2 with openvswitch agent

Considered:

- networking-l2gw driver leveraging bagpipe-bgp running on a ToR
- L3 plugin for inter-subnet distributed routing

2.3 BaGPipe-BGP

BaGPipe-BGP is a component of networking-bagpipe, used on compute nodes along the Neutron agent and bagpipe agent extension of this agent.

It is a lightweight implementation of BGP VPNs (IP VPNs and E-VPNs), targeting deployments on compute nodes hosting VMs, in particular for Openstack/KVM platforms.

The goal of BaGPipe-BGP is *not* to fully implement BGP specifications, but only the subset of specifications required to implement IP VPN VRFs and E-VPN EVIs (RFC4364 a.k.a RFC2547bis, RFC7432/draft-ietf-bess-evpn-overlay, and RFC4684).

BaGPipe-BGP is designed to use encapsulations over IP (such as MPLS-over-GRE or VXLAN), and thus does not require the use of LDP. Bare MPLS over Ethernet is also supported and can be used if compute nodes/routers have direct Ethernet connectivity.

2.3.1 Typical Use/deployment

BaGPipe-BGP has been designed to provide VPN (IP VPN or E-VPN) connectivity to local VMs running on an Openstack compute node.

BaGPipe-BGP is typically driven via its HTTP REST interface, by Openstack Neutron agent extensions found in this package.

Moreover, BaGPipe-BGP can also be used standalone (in particular for testing purposes), with for instance VMs tap interfaces or with veth interfaces to network namespaces (see *below*).

2.3.2 BGP and Route Reflection

If you only want to test how to interconnect one compute node running bagpipe-bgp and an IP/MPLS router, you dont need to setup a BGP Route Reflector.

However, using BaGPipe-BGP between compute nodes currently requires setting up a BGP Route Reflector (see *BGP Implementation* and *Caveats*). Typically, passive mode will have to be used for BGP peerings.

The term BGP Route Reflector refers to a BGP implementation that redistributes routes between iBGP peers RFC4456.

When using bagpipe-bgp on more than one compute node, we thus need each instance of BaGPipe-BGP to be configured to peer with at least one route reflector (see *Configuration*).

We provide a tool that can be used to emulate a route reflector to interconnect **2** BaGPipe-BGP implementations, typically for test purposes (see *Fake RR*).

For more than 2 compute nodes running BaGPipe-BGP, you will need a real BGP implementation supporting RFC4364 and BGP route reflection (and ideally also RFC4684), different options can be considered:

- BGP implementations in other opensource projects would possibly be suitable, but we did not explore these exhaustively:
  - GoBGP , see sample configuration and GoBGP as a RR for bagpipe-bgp PE implementations, with E-VPN
  - we have successfully used OpenBSD BGPd as an IP VPN RR for bagpipe-bgp
  - FRRouting
  - Quagga
- A commercial router from for instance, Alcatel-Lucent, Cisco or Juniper can be used; some of these vendors also provide their OSes as virtual machines

2.3.3 Configuration

The bagpipe-bgp config file default location is: `/etc/bagpipe-bgp/bgp.conf`.

It needs to be customized, at least for the following:

- `local_address`: the local address to use for BGP sessions and traffic encapsulation (can also be specified as an interface, e.g. eth0, in which the IPv4 address of this interface will be used)
- `peers`: the list of BGP peers, it depends on the BGP setup that you have chosen (see above *BGP Route Reflection*)
- dataplane configuration, if you really want packets to get through (see *Dataplane configuration*)

Example with two compute nodes and relying on bagpipe fake route reflector:

- On compute node A (local_address=10.0.0.1):
  - run bagpipe-fakerr
  - run bagpipe-bgp with peers=127.0.0.1 (compute node A will thus connect to the locally running fake route-reflector)
- On compute node B (local_address=10.0.0.2):
  - run bagpipe-bgp with peers=10.0.0.1

Dataplane driver configuration

Note well that the dataplane drivers proposed in the sample config file are *dummy* drivers that will **not** actually drive any dataplane state. To have traffic really forwarded into IP VPNs or E-VPNs, you need to select real dataplane drivers.

For instance, you can use the `ovs` dataplane driver for IP VPN, and the `linux` driver for E-VPN.

Note well that there are specific constraints or dependencies applying to dataplane drivers for IP VPNs:

- the `ovs` driver can be used on most recent Linux kernels, but requires an OpenVSwitch with suitable MPLS code (OVS 2.4 to 2.6 was tested); this driver can do bare-MPLS or MPLS-over-GRE (but see *Caveats* for MPLS-over-GRE); for bare MPLS, this driver requires the OVS bridge to be associated with an IP address, and that VRF interfaces be plugged into OVS prior to calling BaGPipe-BGP API to attach them
- the `linux` driver relies on the native MPLS stack of the Linux kernel, it currently requires a kernel 4.4+ and uses the pyroute2 module that allows defining all states via Netlink rather than by executing ip commands

For E-VPN, the `linux` driver is supported without any particular additional configuration being required, and simply requires a Linux kernel >=3.10 (linux_vxlan.py).

2.3.4 Usage

BaGPipe-BGP local service

If systemd init scripts are installed (see `samples/systemd`), `bagpipe-bgp` is typically started with: `systemctl start bagpipe-bgp`

It can also be started directly with the `bagpipe-bgp` command (`--help` to see what parameters can be used).

By default, it outputs logs on stdin (captured by systemd if run under systemd).

BaGPipe Fake BGP Route Reflector

If you choose to use our fake BGP Route Reflector (see *BGP Route Reflection*), you can start it whether with the `bagpipe-fakerr` command, or if you have startup scripts installed, with `service bagpipe-fakerr start`. Note that this tool requires the additional installation of the `twisted` python package.

There isnt anything to configure, logs will be in syslog.

This tool is not a BGP implementation and simply plugs together two TCP connections face to face.

REST API tool for interface attachments

The `bagpipe-rest-attach` tool allows to exercise the REST API through the command line to attach and detach interfaces from IP VPN VRFs and E-VPN EVIs.

See `bagpipe-rest-attach --help`.

IP VPN example with a VM tap interface

This example assumes that there is a pre-existing tap interface tap42.

- on compute node A, plug tap interface tap42, MAC de:ad:00:00:be:ef, IP 11.11.11.1 into an IP VPN VRF with route-target 64512:77:

  ```
  bagpipe-rest-attach --attach --port tap42 --mac de:ad:00:00:be:ef --ip 11.
  ↪11.11.1 --gateway-ip 11.11.11.254 --network-type ipvpn --rt 64512:77
  ```

- on compute node B, plug tap interface tap56, MAC ba:d0:00:00:ca:fe, IP 11.11.11.2 into an IP VPN VRF with route-target 64512:77:

  ```
  bagpipe-rest-attach --attach --port tap56 --mac ba:d0:00:00:ca:fe --ip 11.
  ↪11.11.2 --gateway-ip 11.11.11.254 --network-type ipvpn --rt 64512:77
  ```

Note that this example is a schoolbook example only, but does not actually work unless you try to use one of the two MPLS Linux dataplane drivers.

Note also that, assuming that VMs are behind these tap interfaces, these VMs will need to have proper IP configuration. When BaGPipe-BGP is use standalone, no DHCP service is provided, and the IP configuration will have to be static.

Another IP VPN example

In this example, the bagpipe-rest-attach tool will build for you a network namespace and a properly configured pair of veth interfaces, and will plug one of the veth to the VRF:

- on compute node A, plug a netns interface with IP 12.11.11.1 into a new IP VPN VRF named test, with route-target 64512:78

  ```
  bagpipe-rest-attach --attach --port netns --ip 12.11.11.1 --network-type␣
  ↪ipvpn --vpn-instance-id test --rt 64512:78
  ```

- on compute node B, plug a netns interface with IP 12.11.11.2 into a new IP VPN VRF named test, with route-target 64512:78

  ```
  bagpipe-rest-attach --attach --port netns --ip 12.11.11.2 --network-type␣
  ↪ipvpn --vpn-instance-id test --rt 64512:78
  ```

For this last example, assuming that you have configured bagpipe-bgp to use the `ovs` dataplane driver for IP VPN, you will actually be able to have traffic exchanged between the network namespaces:

```
ip netns exec test ping 12.11.11.2
PING 12.11.11.2 (12.11.11.2) 56(84) bytes of data.
64 bytes from 12.11.11.2: icmp_req=6 ttl=64 time=1.08 ms
64 bytes from 12.11.11.2: icmp_req=7 ttl=64 time=0.652 ms
```

An E-VPN example

In this example, similarly as the previous one, the bagpipe-rest-attach tool will build for you a network namespace and a properly configured pair of veth interfaces, and will plug one of the veth to the E-VPN instance:

- on compute node A, plug a netns interface with IP 12.11.11.1 into a new E-VPN named test2, with route-target 64512:79

  ```
  bagpipe-rest-attach --attach --port netns --ip 12.11.11.1 --network-type␣
  ↪evpn --vpn-instance-id test2 --rt 64512:79
  ```

- on compute node B, plug a netns interface with IP 12.11.11.2 into a new E-VPN named test2, with route-target 64512:79

  ```
  bagpipe-rest-attach --attach --port netns --ip 12.11.11.2 --network-type␣
  ↪evpn --vpn-instance-id test2 --rt 64512:79
  ```

For this last example, assuming that you have configured bagpipe-bgp to use the `linux` dataplane driver for E-VPN, you will actually be able to have traffic exchanged between the network namespaces:

```
ip netns exec test2 ping 12.11.11.2
PING 12.11.11.2 (12.11.11.2) 56(84) bytes of data.
64 bytes from 12.11.11.2: icmp_req=1 ttl=64 time=1.71 ms
64 bytes from 12.11.11.2: icmp_req=2 ttl=64 time=1.06 ms
```

Looking glass

The REST API (default port 8082) provide troubleshooting information, in read-only, through the /looking-glass URL.

It can be accessed with a browser: e.g. http://10.0.0.1:8082/looking-glass or http://127.0.0.1:8082/looking-glass (a browser extension to nicely display JSON data is recommended).

It can also be accessed with the `bagpipe-looking-glass` utility:

```
# bagpipe-looking-glass
bgp:  (...)
vpns:  (...)
config:  (...)
logs:  (...)
summary:
  warnings_and_errors: 2
  start_time: 2014-06-11 14:52:32
  local_routes_count: 1
  BGP_established_peers: 0
  vpn_instances_count: 1
  received_routes_count: 0
```

```
# bagpipe-looking-glass bgp peers
* 192.168.122.1 (...)
  state: Idle
```

```
# bagpipe-looking-glass bgp routes
match:IPv4/mpls-vpn,*:
  * RD:192.168.122.101:1 12.11.11.1/32 MPLS:[129-B]:
      attributes:
        next_hop: 192.168.122.101
        extended_community: target:64512:78
      afi-safi: IPv4/mpls-vpn
      source: VRF 1 (...)
      route_targets:
        * target:64512:78
match:IPv4/rtc,*:
  * RTC<64512>:target:64512:78:
      attributes:
        next_hop: 192.168.122.101
      afi-safi: IPv4/rtc
      source: BGPManager (...)
match:L2VPN/evpn,*: -
```

2.3.5 Design overview

The main components of BaGPipe-BGP are:

- the engine dispatching events related to BGP routes between workers
- a worker for each BGP peers
- a VPN manager managing the life-cycle of VRFs, EVIs
- a worker for each IP VPN VRF, or E-VPN EVI
- a REST API:
  - to attach/detach interfaces to VRFs and control the parameters for said VRFs
  - to access internal information useful for troubleshooting (/looking-glass/ URL sub-tree)

Publish/Subscribe design

The engine dispatching events related to BGP routes is designed with a publish/subscribe pattern based on the principles in RFC4684. Workers (a worker can be a BGP peer or a local worker responsible for an IP VPN VRF) publish BGP VPN routes with specified Route Targets, and subscribe to the Route Targets that they need to receive. The engine takes care of propagating advertisement and withdrawal events between the workers, based on subscriptions and BGP semantics (e.g. no redistribution between BGP peers sessions).

Best path selection

The core engine does not do any BGP best path selection. For routes received from external BGP peers, best path selection happens in the VRF workers. For routes that local workers advertise, no best path selection is done because two distinct workers will never advertise a route of same BGP NLRI.

Multi-threading

For implementation convenience, the design choice was made to use Python native threads and python Queues to manage the API, local workers, and BGP peers workloads:

- the engine (RouteTableManager) is running as a single thread
- each local VPN worker has its own thread to process route events
- each BGP peer worker has two threads to process outgoing route events, and receive socket data, plus a few timers.
- VPN port attachment actions are done in the main thread handling initial setup and API calls, these calls are protected by Python locks

Non-persistency of VPN and port attachments

The BaGPipe-BGP service, as currently designed, does not persist information on VPNs (VRFs or EVIs) and the ports attached to them. On a restart, the component responsible triggering the attachment of interfaces to VPNs, can detect the restart of the BGP and re-trigger these attachments.

BGP Implementation

The BGP protocol implementation reuses BGP code from ExaBGP. BaGPipe-BGP only reuses the low-level classes for message encodings and connection setup.

Non-goals for this BGP implementation:

- full-fledged BGP implementation
- redistribution of routes between BGP peers (hence, no route reflection, no eBGP)
- accepting incoming BGP connections
- scaling to a number of routes beyond the number of routes required to route traffic in/out of VMs hosted on a compute node running BaGPipe-BGP

Dataplanes

BaGPipe-BGP was designed to allow for a modular dataplane implementation. For each type of VPN (IP VPN, E-VPN) a dataplane driver is chosen through configuration. A dataplane driver is responsible for setting up forwarding state for incoming and outgoing traffic based on port attachment information and BGP routes.

(see *Dataplane driver configuration*)

2.3.6 Caveats

- BGP implementation not written for compliancy
  - the BaGPipe-BGP service does not listen for incoming BGP connections (using a BGP route reflector is required to interconnect bagpipe-bgp instance together, typically using passive mode for BGP peerings)
  - the state machine, in particular retry timers is possibly not fully compliant
  - however, interop testing has been done with a fair amount of implementations
- standard MPLS-over-GRE, interoperating with routers, requires OVS >= 2.8 (previous Open-VSwitch releases do MPLS-o-Ethernet-o-GRE and not MPLS-o-GRE)

CHAPTER

THREE

INSTALLATION

3.1 Networking-bagpipe installation

The details related to how a package should be installed may depend on your environment.

If possible, you should rely on packages provided by your Linux and/or OpenStack distribution.

If you use `pip`, follow these steps to install networking-bagpipe:

- identify the version of the networking-bagpipe package that matches your Openstack version:
  - Liberty: most recent of 3.0.x
  - Mitaka: most recent of 4.0.x
  - Newton: most recent of 5.0.x
  - Ocata: most recent of 6.0.x
  - Pike: most recent of 7.0.x
  - Queens: most recent of 8.0.x
  - (see https://releases.openstack.org/index.html)
- indicate pip to (a) install precisely this version and (b) take into account Openstack upper constraints on package versions for dependencies (example for Queens):

  ```
  $ pip install -c https://releases.openstack.org/constraints/upper/queens
  ```

3.2 BaGPipe for Neutron L2

3.2.1 Installation in a devstack test/development environment

- install devstack (whether stable/**x** or master)
- enable the devstack plugin by adding this to `local.conf`:
  - to use branch `stable/x` (e.g. *stable/queens*):

    ```
    enable_plugin networking-bagpipe https://git.openstack.org/openstack/
    ↪networking-bagpipe.git stable/X
    ```

  - to use the development branch:

```
enable_plugin networking-bagpipe https://git.openstack.org/openstack/
↪networking-bagpipe.git master
```

- enable bagpipe ML2 by adding this to `local.conf`:

```
ENABLE_BAGPIPE_L2=True
```

- for multinode setups, configure *BaGPipe-BGP* on each compute node, i.e. you need each *BaGPipe-BGP* to peer with a BGP Route Reflector:
  - in `local.conf`:

    ```
    # IP of your route reflector or BGP router, or fakeRR:
    BAGPIPE_BGP_PEERS=1.2.3.4
    ```

  - for two compute nodes, you can use the FakeRR provided in *BaGPipe-BGP*
  - for more than two compute nodes, you can use GoBGP (sample configuration) or a commercial E-VPN implementation (e.g. vendors participating in EANTC interop testing on E-VPN)

3.2.2 Deployment

On Neutron servers, the following needs to be done, *based on an ML2/linuxbridge or ML2/openvswitch configuration* as a starting point:

- installing `networking-bagpipe` python package (see *Networking-bagpipe installation*)
- in ML2 configuration (`/etc/neutron/plugins/ml2.ini`):
  - adding the `bagpipe` mechanism driver (additionally to the `linuxbridge` or `openvswitch` driver which will still handle `flat` and `vlan` networks)
  - *before Queens release* (i.e. if networking-bagpipe < 8) use the `route_target` type driver as default
  - result:

    ```
    [ml2]
    # tenant_network_types = route_target  # before queens only!
    mechanism_drivers = openvswitch,linuxbridge,bagpipe
    ```

You need to deploy a BGP Route Reflector, that will distribute BGP VPN routes among compute and network nodes. This route reflector will need to support E-VPN and, optionally, RT Constraints. One option, among others is to use GoBGP (sample configuration).

On compute node (and network nodes if any) the following needs to be done, *based on an ML2/linuxbridge or ML2/openvswitch configuration* as a starting point:

- installing `networking-bagpipe` python package (see *Networking-bagpipe installation*)
- configuring Neutron linuxbridge or OpenvSwitch agent for bagpipe `/etc/neutron/plugins/ml2.ini`:
  - enabling `bagpipe` agent extension
  - *before Queens release* (i.e. if networking-bagpipe < 8), disable VXLAN:

  - configuring the AS number and range to use to allocate BGP Route Targets for tenant networks
  - result:

```
[agent]
extensions = bagpipe

[vxlan]
# for a release strictly before OpenStack Queens (networking-bagpipe
↪< 8)
# enable_vxlan = False

[ml2_bagpipe_extension]
as_number = 64512
```

- configuring *BaGPipe-BGP*:
  - setting `local_address` to the compute node address (or the name of one of its interfaces e.g. eth0)
  - adding the Route Reflector IP to `peers`
  - selecting the EVPN dataplane driver corresponding to your agent in (`/etc/bagpipe-bgp/bgp.conf`):
    * `ovs` for the openvswitch agent:

```
[DATAPLANE_DRIVER_EVPN]
dataplane_driver = ovs
```

    * `linux` for the linuxbridge agent:

```
[DATAPLANE_DRIVER_EVPN]
dataplane_driver = linux
```

3.3 BaGPipe for BGPVPN

Information on how to use `bagpipe` driver for networking-bgpvpn is provided in BGPVPN bagpipe driver documentation.

3.4 BaGPipe for networking-sfc

To enable the use of networking-bagpipe driver for networking-sfc, the following needs to be done:

- enable `bagpipe` driver for the `networking-sfc` service plugin, in `/etc/neutron/neutron.conf` and configure its parameters (see *SFC*):

```
[sfc]
drivers = bagpipe
```

(continues on next page)

(continued from previous page)

```
[sfc_bagpipe]
# examples, of course!
as_number = 64517
rtnn = 10000,30000
```

- add the `bagpipe_sfc` agent extension to the Neutron linuxbridge agent config in“/etc/neutron/plugins/ml2.ini“:

```
[agent]
extensions = bagpipe_sfc
```

- *BaGPipe-BGP* lightweight BGP VPN implementation, configured to use `ovs` as dataplane driver for IPVPNs, and `linux` as dataplane driver for EVPN (`/etc/bagpipe-bgp/bgp.conf`):

```
[DATAPLANE_DRIVER_IPVPN]
dataplane_driver = ovs

[DATAPLANE_DRIVER_EVPN]
dataplane_driver = linux
```

3.4.1 In a devstack

To experiment with sfc driver in a devstack, the following is can be added in your *local.conf* (replace stable/X with stable/queens for e.g. Openstack Queens release) :

```
enable_plugin networking-sfc https://git.openstack.org/openstack/
↪networking-bagpipe.git
# enable_plugin networking-sfc https://git.openstack.org/openstack/
↪networking-bagpipe.git stable/X
enable_plugin networking-bagpipe https://git.openstack.org/openstack/
↪networking-bagpipe.git
# enable_plugin networking-bagpipe https://git.openstack.org/
↪openstack/networking-bagpipe.git stable/X

BAGPIPE_DATAPLANE_DRIVER_EVPN=linux
BAGPIPE_DATAPLANE_DRIVER_IPVPN=ovs

[[post-config|$NEUTRON_CONF]]

[sfc]
drivers = bagpipe

[sfc_bagpipe]
as_number = 64517
rtnn = 10000,30000


[[post-config|/$NEUTRON_CORE_PLUGIN_CONF]]

```

(continues on next page)

(continued from previous page)

```
[agent]
extensions = bagpipe_sfc
```

CHAPTER

FOUR

CONFIGURATION OPTIONS

This section provides a list of all possible options for each configuration file.

4.1 Configuration Reference

networking-bagpipe uses the following configuration files for its various services.

4.1.1 Neutron config

SFC

The following section can be added to Neutron server configuration to parameters related to the sfc driver.

sfc_bagpipe

as_number

Type
integer

Default
`64512`

Autonomous System number used to generate BGP Route Targets that will be used for Port Chain allocations

rtnn

Type
list

Default
`[5000, 5999]`

List containing <rtnn_min>, <rtnn_max> defining a range of BGP Route Targets that will be used for Port Chain allocations. This range MUST not intersect the one used for network segmentation identifiers

4.1.2 Neutron agent config

The following section can be added to Neutron agent configuration.

bagpipe

`mpls_bridge`

Type
string

Default
`br-mpls`

OVS MPLS bridge to use

bagpipe_ml2_extension

`as_number`

Type
list

Default
`[64512]`

Autonomous System number used to generate BGP RTs forE-VPNs used by bagpipe ML2 (more than one is possible,to allow a deployment to do a 2-step transition to change the AS number used)

4.1.3 bagpipe-bgp.conf

api

`host`

Type
host address

Default
`127.0.0.1`

IP address on which the API server should listen

Table 1: Deprecated Variations

Group	Name
api	api_host

`port`

Type
port number

Default
`8082`

Minimum Value
0

Maximum Value
65535

Port on which the API server should listen

Table 2: Deprecated Variations

Group	Name
api	api_port

bgp

`local_address`

Type
unknown type

Default
`<None>`

IP address used for BGP peerings

`peers`

Type
list

Default
`[]`

IP addresses of BGP peers

`my_as`

Type
integer

Default
`<None>`

Minimum Value
1

Maximum Value
65535

Our BGP Autonomous System

`enable_rtc`

Type
boolean

Default
`True`

Enable RT Constraint (RFC4684)

`bgp_port`

Type
port number

Default
`179`

Minimum Value
0

Maximum Value
65535

TCP port of connections to BGP peers

common

`root_helper`

Type
string

Default
`sudo`

Root helper command.

`root_helper_daemon`

Type
string

Default
`<None>`

Root helper daemon application to use when possible.

dataplane_driver_evpn

`dataplane_local_address`

Type
unknown type

Default
`<None>`

IP address to use as next-hop in our route advertisements, will be used to send us VPN traffic

`dataplane_driver`

Type
string

Default
`dummy`

Dataplane driver.

dataplane_driver_ipvpn

`dataplane_local_address`

Type
unknown type

Default
`<None>`

IP address to use as next-hop in our route advertisements, will be used to send us VPN traffic

`dataplane_driver`

Type
string

Default
`dummy`

Dataplane driver.

More dataplane configuration parameters exist depending on the driver:

[DATAPLANE_DRIVER_EVPN] with driver=linux

dataplane_driver_evpn

`vxlan_dst_port`

Type
integer

Default
`4789`

UDP port toward which send VXLAN traffic (defaults to standard IANA-allocated port)

[DATAPLANE_DRIVER_IPVPN] with driver=linux

dataplane_driver_ipvpn

`mpls_interface`

Type
string

Default
`<None>`

Interface used to send/receive MPLS traffic. Use *gre* to choose automatic creation of a tunnel interface for MPLS/GRE encap

[DATAPLANE_DRIVER_IPVPN] with driver=ovs

dataplane_driver_ipvpn

`mpls_interface`

Type
string

Default
`<None>`

Interface used to send/receive MPLS traffic. Use *gre* to choose automatic creation of a tunnel port for MPLS/GRE encap

`mpls_over_gre`

Type
string

Default
`auto`

Valid Values
auto, True, False

Advanced Option
Intended for advanced users and not used by the majority of users, and might have a significant effect on stability and/or performance.

Force the use of MPLS/GRE even with mpls_interface specified

`proxy_arp`

Type
boolean

Default
`False`

Advanced Option
Intended for advanced users and not used by the majority of users, and might have a significant effect on stability and/or performance.

Activate ARP responder per VRF for any IP address

`arp_responder`

Type
boolean

Default
`False`

Advanced Option
Intended for advanced users and not used by the majority of users, and might have a significant effect on stability and/or performance.

ARP responder per VRF

`vxlan_encap`

Type
boolean

Default
`False`

Advanced Option
Intended for advanced users and not used by the majority of users, and might have a significant effect on stability and/or performance.

Be ready to receive VPN traffic as VXLAN, and to preferrably send traffic as VXLAN when advertised by the remote end

`ovs_bridge`

Type
string

Default
`br-mpls`

Advanced Option
Intended for advanced users and not used by the majority of users, and might have a significant effect on stability and/or performance.

`input_table`

Type
integer

Default
`0`

Advanced Option
Intended for advanced users and not used by the majority of users, and might have a significant effect on stability and/or performance.

`ovs_table_id_start`

Type
integer

Default
`1`

Advanced Option
Intended for advanced users and not used by the majority of users, and might have a significant effect on stability and/or performance.

`gre_tunnel`

Type
string

Default
`mpls_gre`

Advanced Option
Intended for advanced users and not used by the majority of users, and might have a significant effect on stability and/or performance.

OVS interface name for MPLS/GRE encap

`gre_tunnel_options`

Type
list

Default
`[]`

Options, comma-separated, passed to OVS for GRE tunnel port creation (e.g. packet_type=legacy_l3,) that will be added as OVS tunnel interface options (e.g. options:packet_type=legacy_l3 options:)

`ovsbr_interfaces_mtu`

Type
integer

Default
`<None>`

Advanced Option
Intended for advanced users and not used by the majority of users, and might have a significant effect on stability and/or performance.

`hash_method`

Type
string

Default
`dp_hash`

Valid Values
hash, dp_hash

Advanced Option
Intended for advanced users and not used by the majority of users, and might have a significant effect on stability and/or performance.

Can be used to control the OVS group bucket selection method (mapped to ovs selection_method)

`hash_method_param`

Type
string

Default
`0`

Advanced Option
Intended for advanced users and not used by the majority of users, and might have a significant effect on stability and/or performance.

Can be used to control the OVS group bucket selection method (mapped to ovs selection_method_param)

`hash_fields`

Type
list

Default
`[]`

Advanced Option
Intended for advanced users and not used by the majority of users, and might have a significant effect on stability and/or performance.

Can be used to control the fields used by the OVS group bucket selection method (mapped to ovs fields)

4.2 Sample Configuration Files

The following are sample configuration files for all networking-bagpipe. These are generated from code and reflect the current state of code in the networking-bagpipe repository.

4.2.1 Sample Neutron agent config

This sample configuration can also be viewed in the raw format.

```
[DEFAULT]


[bagpipe]

#
# From networking_bagpipe.bagpipe_bgp_agent
#

```

(continues on next page)

(continued from previous page)

```
# OVS MPLS bridge to use (string value)
#mpls_bridge = br-mpls


[bagpipe_ml2_extension]

#
# From networking_bagpipe.bagpipe_bgp_agent
#

# Autonomous System number used to generate BGP RTs forE-VPNs used by bagpipe
# ML2 (more than one is possible,to allow a deployment to do a 2-step
# transition to change the AS number used) (list value)
#as_number = 64512
```

4.2.2 Sample bagpipe-bgp.conf

This sample configuration can also be viewed in the raw format.

```
[DEFAULT]


[api]

#
# From networking_bagpipe.api
#

# IP address on which the API server should listen (host address value)
# Deprecated group/name - [api]/api_host
#host = 127.0.0.1

# Port on which the API server should listen (port value)
# Minimum value: 0
# Maximum value: 65535
# Deprecated group/name - [api]/api_port
#port = 8082


[bgp]

#
# From networking_bagpipe.bgp_common
#

# IP address used for BGP peerings (interface address value)
#local_address = <None>
```

(continues on next page)

(continued from previous page)

```
# IP addresses of BGP peers (list value)
#peers =

# Our BGP Autonomous System (integer value)
# Minimum value: 1
# Maximum value: 65535
#my_as = <None>

# Enable RT Constraint (RFC4684) (boolean value)
#enable_rtc = true

# TCP port of connections to BGP peers (port value)
# Minimum value: 0
# Maximum value: 65535
#bgp_port = 179


[common]

#
# From networking_bagpipe.run_command
#

# Root helper command. (string value)
#root_helper = sudo

# Root helper daemon application to use when possible. (string value)
#root_helper_daemon = <None>


[dataplane_driver_evpn]

#
# From networking_bagpipe.dataplane.evpn
#

# IP address to use as next-hop in our route advertisements, will be used to
# send us VPN traffic (interface address value)
#dataplane_local_address = <None>

# Dataplane driver. (string value)
#dataplane_driver = dummy


[dataplane_driver_ipvpn]

#
# From networking_bagpipe.dataplane.ipvpn
#
```

(continues on next page)

(continued from previous page)

```
# IP address to use as next-hop in our route advertisements, will be used to
# send us VPN traffic (interface address value)
#dataplane_local_address = <None>

# Dataplane driver. (string value)
#dataplane_driver = dummy
```

More dataplane configuration parameters exist depending on the driver:

Sample [DATAPLANE_DRIVER_EVPN] with driver=linux

This sample configuration can also be viewed in the raw format.

```
[DEFAULT]


[dataplane_driver_evpn]

#
# From networking_bagpipe.dataplane.evpn.linux_vxlan
#

# UDP port toward which send VXLAN traffic (defaults to standard IANA-
↪allocated
# port) (integer value)
#vxlan_dst_port = 4789
```

Sample [DATAPLANE_DRIVER_IPVPN] with driver=linux

This sample configuration can also be viewed in the raw format.

```
[DEFAULT]


[dataplane_driver_ipvpn]

#
# From networking_bagpipe.dataplane.ipvpn.mpls_linux
#

# Interface used to send/receive MPLS traffic. Use '*gre*' to choose automatic
# creation of a tunnel interface for MPLS/GRE encap (string value)
#mpls_interface = <None>
```

Sample [DATAPLANE_DRIVER_IPVPN] with driver=ovs

This sample configuration can also be viewed in the raw format.

```
[DEFAULT]


[dataplane_driver_ipvpn]

#
# From networking_bagpipe.dataplane.ipvpn.mpls_ovs
#

# Interface used to send/receive MPLS traffic. Use '*gre*' to choose automatic
# creation of a tunnel port for MPLS/GRE encap (string value)
#mpls_interface = <None>

# Options, comma-separated, passed to OVS for GRE tunnel port creation (e.g.
# 'packet_type=legacy_l3, ...') that will be added as OVS tunnel interface
# options (e.g. 'options:packet_type=legacy_l3 options:...') (list value)
#gre_tunnel_options =

# Force the use of MPLS/GRE even with mpls_interface specified (string value)
# Possible values:
# auto - <No description provided>
# True - <No description provided>
# False - <No description provided>
# Advanced Option: intended for advanced users and not used
# by the majority of users, and might have a significant
# effect on stability and/or performance.
#mpls_over_gre = auto

# Activate ARP responder per VRF for any IP address (boolean value)
# Advanced Option: intended for advanced users and not used
# by the majority of users, and might have a significant
# effect on stability and/or performance.
#proxy_arp = false

# ARP responder per VRF (boolean value)
# Advanced Option: intended for advanced users and not used
# by the majority of users, and might have a significant
# effect on stability and/or performance.
#arp_responder = false

# Be ready to receive VPN traffic as VXLAN, and to preferrably send traffic as
# VXLAN when advertised by the remote end (boolean value)
# Advanced Option: intended for advanced users and not used
# by the majority of users, and might have a significant
# effect on stability and/or performance.
#vxlan_encap = false

```

(continues on next page)

(continued from previous page)

```
# (string value)
# Advanced Option: intended for advanced users and not used
# by the majority of users, and might have a significant
# effect on stability and/or performance.
#ovs_bridge = br-mpls

# (integer value)
# Advanced Option: intended for advanced users and not used
# by the majority of users, and might have a significant
# effect on stability and/or performance.
#input_table = 0

# (integer value)
# Advanced Option: intended for advanced users and not used
# by the majority of users, and might have a significant
# effect on stability and/or performance.
#ovs_table_id_start = 1

# OVS interface name for MPLS/GRE encap (string value)
# Advanced Option: intended for advanced users and not used
# by the majority of users, and might have a significant
# effect on stability and/or performance.
#gre_tunnel = mpls_gre

# (integer value)
# Advanced Option: intended for advanced users and not used
# by the majority of users, and might have a significant
# effect on stability and/or performance.
#ovsbr_interfaces_mtu = <None>

# Can be used to control the OVS group bucket selection method (mapped to ovs
# 'selection_method') (string value)
# Possible values:
# hash - <No description provided>
# dp_hash - <No description provided>
# Advanced Option: intended for advanced users and not used
# by the majority of users, and might have a significant
# effect on stability and/or performance.
#hash_method = dp_hash

# Can be used to control the OVS group bucket selection method (mapped to ovs
# 'selection_method_param') (string value)
# Advanced Option: intended for advanced users and not used
# by the majority of users, and might have a significant
# effect on stability and/or performance.
#hash_method_param = 0

# Can be used to control the fields used by the OVS group bucket selection
# method (mapped to ovs 'fields') (list value)
```

(continues on next page)

(continued from previous page)

```
# Advanced Option: intended for advanced users and not used
# by the majority of users, and might have a significant
# effect on stability and/or performance.
#hash_fields =
```

CHAPTER

FIVE

DEVELOPMENT

5.1 Contributing

If you would like to contribute to the development of OpenStack, you must follow the steps in this page: https://docs.openstack.org/infra/manual/developers.html

Once those steps have been completed, changes to OpenStack should be submitted for review via the Gerrit tool, following the workflow documented at: https://docs.openstack.org/infra/manual/developers.html#development-workflow

Pull requests submitted through GitHub will be ignored.

Bugs should be filed on Launchpad, not GitHub: https://bugs.launchpad.net/networking-bagpipe