Networking-vpp: An OpenStack ml2 driver for VPP

Jerome Tollet / Ian Wells

FD.io Program | April 30th, 2017

Agenda

- What is networking-vpp?
- Design principles
- Overall architecture
- Current feature set
- Focus on HA features
  - Clustering *etcd*
  - Transactional port setup
- Focus on Security features
- Roadmap for 17.07
- Questions

What is networking-vpp

- FD.io / VPP is a **fast software dataplane** that can be used to speed up communications for any kind of VM or VNF.

- VPP can speed-up both East-West and North-South communications

- Networking-vpp is a project aiming at providing a **simple, robust, production grade** integration of VPP in OpenStack using ml2 interface

- Goal is to make VPP a **first class citizen component in OpenStack** for NFV and Cloud applications

Networking-vpp: Design Principles

- Main design goals are: **simplicity, robustness, scalability**

- Efficient management communications
  - All communication is asynchronous
  - All communication is REST based
- Robustness
  - Built for failure – if a cloud runs long enough, *everything* will happen eventually
  - All modules are unit and system tested
- Code is small and easy to understand (no spaghetti/legacy code)

Networking-vpp: current feature set

- Network types
    - VLAN: supported since version 16.09
    - VXLAN-GPE: supported since version 17.04
- Port types
    - VM connectivity done using fast *vhostuser* interfaces
    - TAP interfaces for services such as DHCP
- Security
    - Security-groups based on VPP stateful ACLs
    - Port Security can be disabled for true fastpath
    - Role Based Access Control and secure TLS connections for etcd
- Layer 3 Networking
    - North-South Floating IP
    - North-South SNAT
    - East-West Internal Gateway
- Robustness
    - If Neutron commits to it, *it will happen*
    - Component state resync in case of failure: recovers from restart of Neutron, the agent and VPP

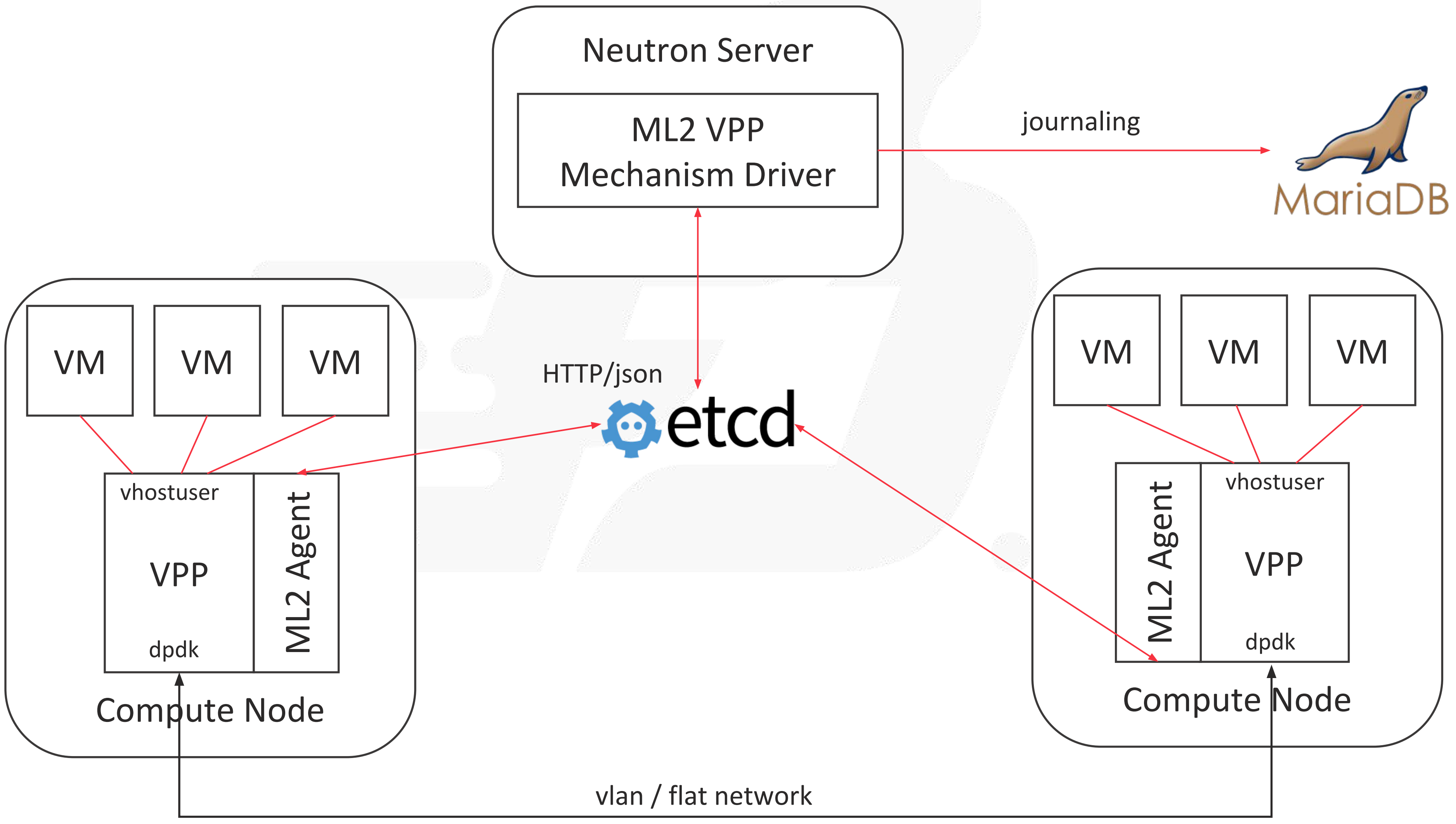
Networking-vpp, what is your problem?

- You have a controller and you tell it to do something
- It talks to a device to get the job done
  - Did the message even get sent, or did the controller crash first? Does the controller believe it sent the message when it restarts?
  - Did the message get sent and the controller crash before it got a reply?
  - Did it get a reply but crash before it processed it?
  - If the message was sent and you got a reply, did the device get programmed?
  - If the message was sent and you *didn't* get a reply, did the device get programmed?

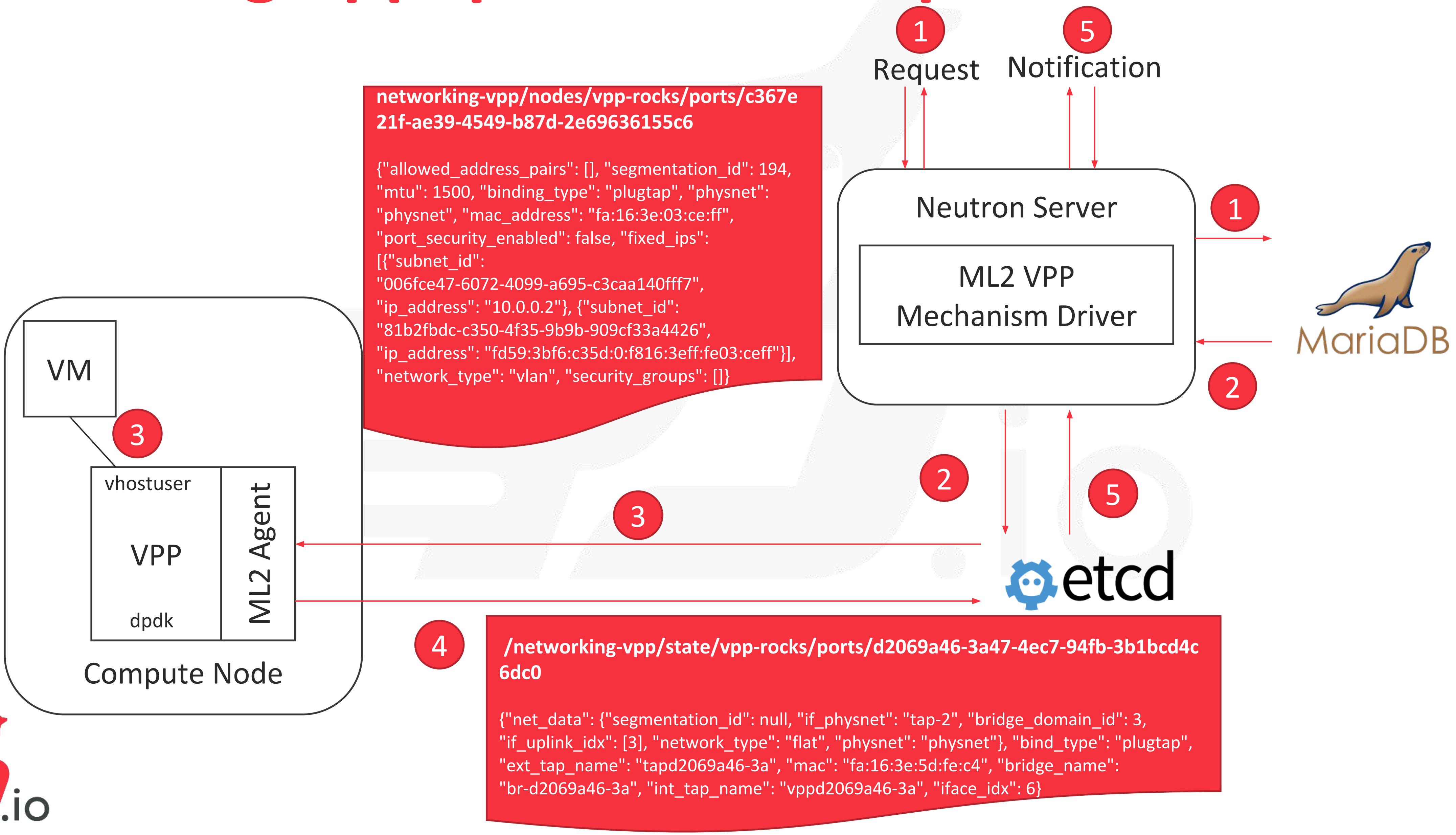
Networking-vpp, what is your problem?

- If you give a device a list of jobs to do, it’s *really hard* to make sure it gets them all and acts on them
- If you give a device a *description of the state you want it to get to*, the task can be made *much* easier

Networking-vpp: overall architecture



Networking-vpp: port creation process



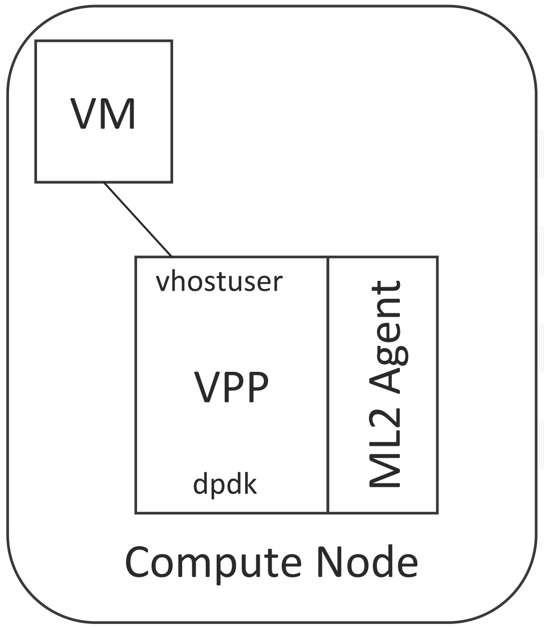
networking-vpp/nodes/vpp-rocks/ports/c367e21f-ae39-4549-b87d-2e69636155c6

{"allowed_address_pairs": [], "segmentation_id": 194, "mtu": 1500, "binding_type": "plugtap", "physnet": "physnet", "mac_address": "fa:16:3e:03:ce:ff", "port_security_enabled": false, "fixed_ips": [{"subnet_id": "006fce47-6072-4099-a695-c3caa140fff7", "ip_address": "10.0.0.2"}, {"subnet_id": "81b2fbdc-c350-4f35-9b9b-909cf33a4426", "ip_address": "fd59:3bf6:c35d:0:f816:3eff:fe03:ceff"}], "network_type": "vlan", "security_groups": []}

/networking-vpp/state/vpp-rocks/ports/d2069a46-3a47-4ec7-94fb-3b1bcd4c6dc0

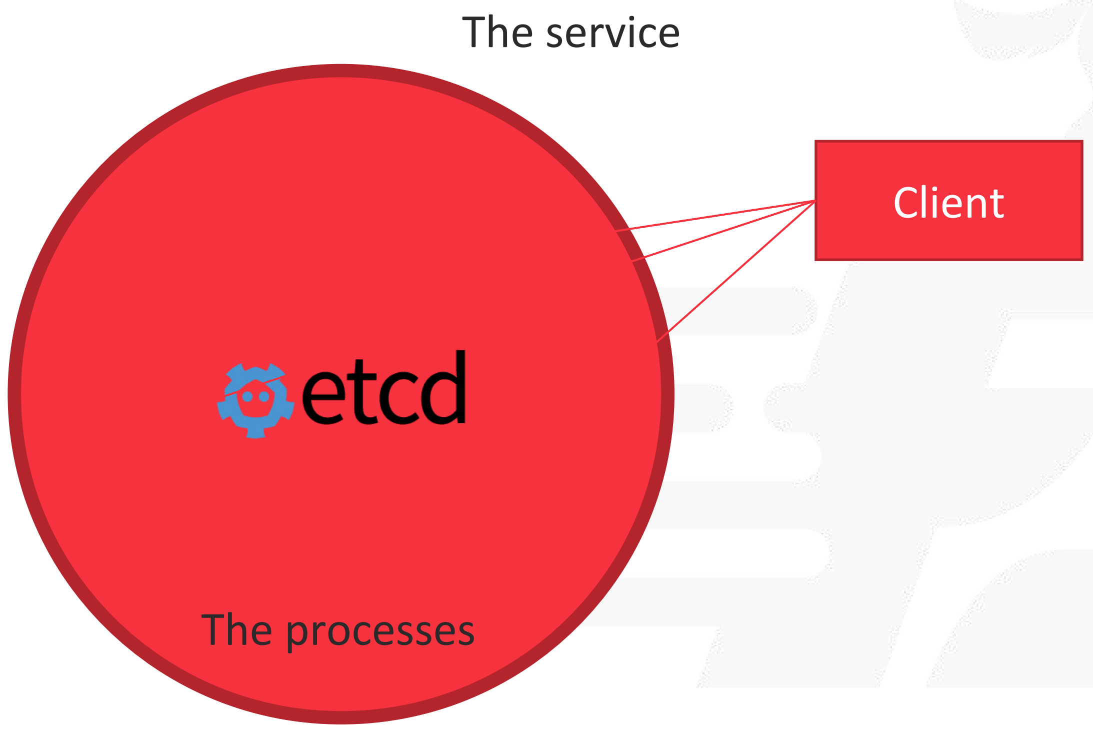
{"net_data": {"segmentation_id": null, "if_physnet": "tap-2", "bridge_domain_id": 3, "if_uplink_idx": [3], "network_type": "flat", "physnet": "physnet"}, "bind_type": "plugtap", "ext_tap_name": "tapd2069a46-3a", "mac": "fa:16:3e:5d:fe:c4", "bridge_name": "br-d2069a46-3a", "int_tap_name": "vppd2069a46-3a", "iface_idx": 6}

Networking-vpp: Resync mechanism



- The agent marks everything it puts in VPP
- If the agent restarts, it comes up with no knowledge of what's in VPP, so it reads those marks back
- While it's been gone, things may have happened and etcd will have changed
- For each item, it can see if it's correct or if it's out of date (for instance, deleted), and it can also spot new ports it's been asked to make
- With that information, it does the minimum necessary to fix VPP's state
- This means that while the agent's gone, traffic keeps moving
- Neutron is abiding by its promises ('I will do this thing for you at the first possible moment')

Networking-vpp: HA etcd deployment



- etcd is a strictly consistent store that can run with multiple processes
- They talk to each other as they get the job done
- A client can talk to any of them
- Various deployment models – the client can use a proxy or talk directly to the etcd processes

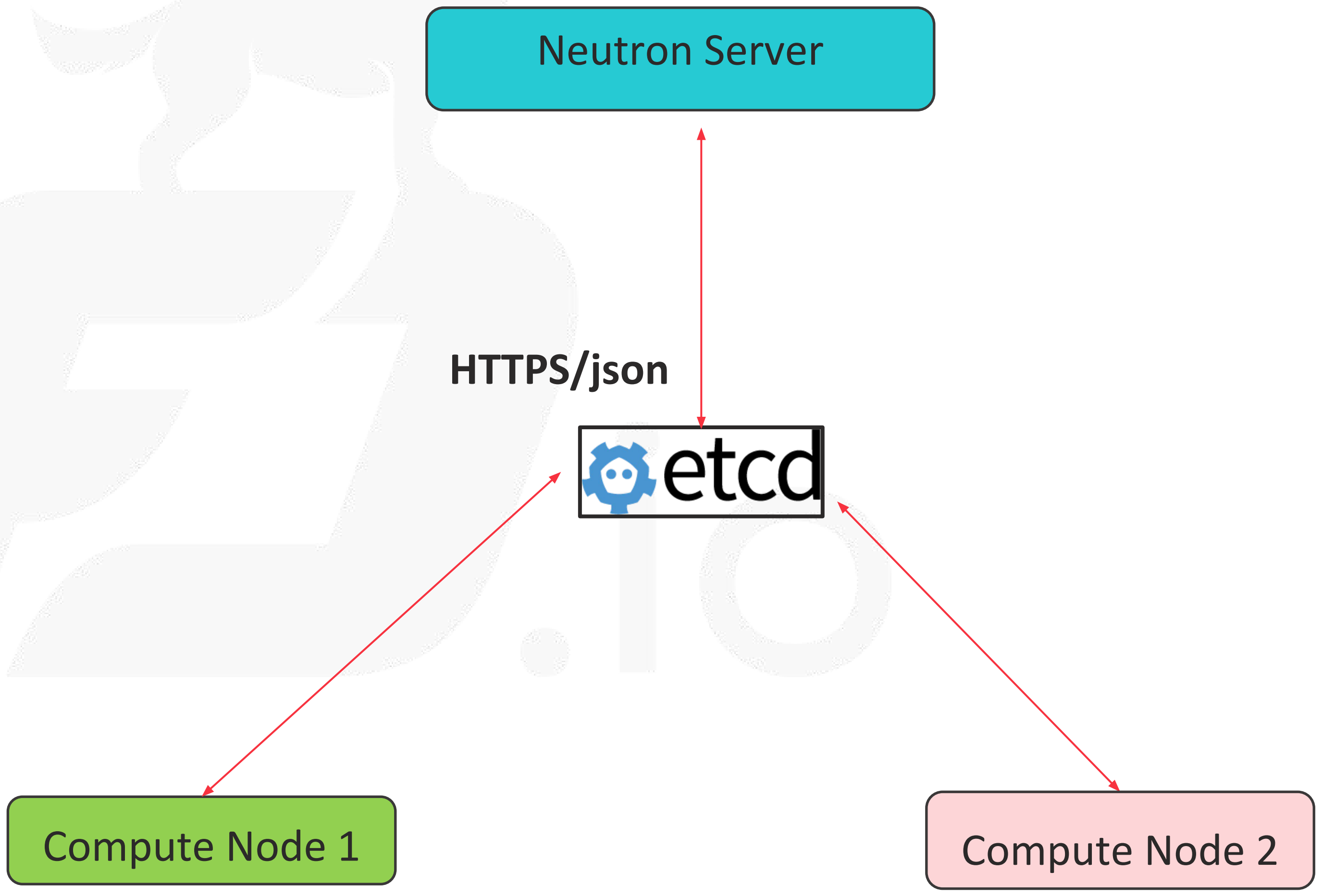
Networking-vpp: Security and RBAC

- Etcd can work over TLS (HTTPS) when talking to itself and its clients
- Each client can have a credential that gives it limited access (R, RW, None)
- Networking-VPP is designed so that nearly all the keys are written by only one type of process

RBAC can be used to protect the datastore from both confused and malicious processes

Networking-vpp: Role Based Access Control

- Security Hardening
  - TLS communication between nodes and ETCD :
    - Confidentiality and integrity of the messages in transit and authentication of ETCD server.
  - ETCD RBAC :
    - Limit the impact of a compromised Compute Node to the node itself



Networking-vpp: Role Based Access Control

	ETCD/Node1 Requests	ETCD/Node1 Status	ETCD/Node2 Requests	ETCD/Node2 Status
Neutron Server	R/W	R/O	R/W	R/O
Compute Node 1	R/O	R/W	Access Denied	Access Denied
Compute Node 2	Access Denied	Access Denied	R/O	R/W

Networking-vpp: simplicity

```
$ rm -rf tests
$ wc `find . -name \*.py -print`
    2886   11520  123320 ./agent/server.py
[...]
     921    2834   35173 ./agent/vpp.py
[...]
    1067    4607   46112 ./mech_vpp.py
[...]
    7336   27864  290940 total
```

Most of the agent code

A nicer VPP Python API

Most of the mechanism driver

NB: this code ***has comments*** – *1839 lines of code* in the above three files

Networking-vpp: extensibility

Today: the agent will implement in VPP whatever the datamodel in etcd says it needs

It can do more:

- Extend that model
- Plug in code to the agent to respond to it

Benefit: keeping all the robustness

Limitation: where do we put the intelligence?

Networking-vpp: Port bind request etcd model

Port state Key/Value sample

- networking-vpp/nodes/{SERVER}/{PORT UUID}

Port state Key/Value sample

networking-vpp/nodes/vpp-rocks/ports/c367e21f-ae39-4549-b87d-2e69636155c6

```
{       "allowed_address_pairs": [],
        "segmentation_id": 194,
        "mtu": 1500,
        "binding_type": "plugtap",
        "physnet": "physnet",
        "mac_address": "fa:16:3e:03:ce:ff",
        "port_security_enabled": false,
        "fixed_ips": [   {"subnet_id": "006fce47-6072-4099-a695-c3caa140fff7", "ip_address": "10.0.0.2"},
                {"subnet_id": "81b2fbdc-c350-4f35-9b9b-909cf33a4426", "ip_address": "fd59:3bf6:c35d:0:f816:3eff:fe03:ceff"}],
        "network_type": "vlan", "security_groups": []
}
```

Networking-vpp: port state etcd model

Port state Key/Value model

- /networking-vpp/state/{SERVER NAME}/ports/{PORT UUID}
- the ports that the VPP agent has programmed VPP with (the same list - it's done all its jobs), plus the physnets that this host knows about
- Key is inserted by the Agent in etcd when port is created into VPP.
    - Value is for debugging purpose (e.g. port indexes, bridge names)

Port state Key/Value sample

/networking-vpp/state/vpp-rocks/ports/d2069a46-3a47-4ec7-94fb-3b1bcd4c6dc0

```
{"net_data": { "segmentation_id": null, "if_physnet": "tap-2", "bridge_domain_id": 3,
        "if_uplink_idx": [3], "network_type": "flat", "physnet": "physnet"},
"bind_type": "plugtap",
"ext_tap_name": "tapd2069a46-3a",
"mac": "fa:16:3e:5d:fe:c4",
"bridge_name": "br-d2069a46-3a",
"int_tap_name": "vppd2069a46-3a",
"iface_idx": 6}
```

Networking-vpp: Security Groups etcd model

Security Group etcd Key/Value structure

- /networking-vpp/global/secgroups/{SECGROUP UUID} -> JSON SecGroup detection

Security Group Key/Value sample

```
/networking-vpp/global/secgroups/2be282a7-0a04-4411-9a8f-4eeb1c676454
{"ingress_rules": [
        {"is_ipv6": 0, "remote_ip_addr": "0.0.0.0", "ip_prefix_len": 0, "protocol": 0, "port_min": 0, "port_max": 0},
        {"is_ipv6": 1, "remote_ip_addr": "0:0:0:0:0:0:0:0", "ip_prefix_len": 0, "protocol": 0, "port_min": 0, "port_max": 0}
"egress_rules": [
        {"is_ipv6": 1, "remote_ip_addr": "0:0:0:0:0:0:0:0", "ip_prefix_len": 0, "protocol": 0, "port_min": 0, "port_max": 0}, {"is_ipv6": 0,
        "remote_ip_addr": "0.0.0.0", "ip_prefix_len": 0, "protocol": 0, "port_min": 0, "port_max": 0}]
}
```

Networking-vpp: [VXLAN|LISP]-GPE

- VXLAN-GPE was introduced in networking-vpp 17.04
- Encapsulate layer 2 frames from the VMs into layer 3 packets (GPE)
- Provides very large horizontal scaling and isolation

- When a VM is created, MAC address (aka EID) is populated in all VPP bridge domains belonging to the same VNI

- GPE data stored under /networking-vpp/global/networks/gpe

Networking-vpp: [VXLAN|LISP]-GPE etcd model

GPE etcd Key/Value structure

- /networking-vpp/global/networks/gpe/{VNI}/{HOSTNAME}/{MAC_ADDRESS} -> Underlay IP address

GPE Key/Value sample

- /networking-vpp/global/networks/gpe/1007/sjo-smf-ubuntu-server-3/fa:16:3e:ba:a8:12
- /networking-vpp/global/networks/gpe/1007/sjo-smf-ubuntu-server-3/fa:16:3e:20:3e:c2
- /networking-vpp/global/networks/gpe/1007/sjo-smf-ubuntu-server-2/fa:16:3e:eb:c9:d8
- /networking-vpp/global/networks/gpe/1007/sjo-smf-ubuntu-server-2/fa:16:3e:82:50:81

Networking-vpp: layer 3 routers model

Layer 3 etcd Key/Value structure

- /networking-vpp/nodes/{SERVER}/routers/interface/{INTF UUID}
  - Details of a single Neutron router interface i.e. tenant network interface on a neutron router
- /networking-vpp/nodes/{SERVER}/routers/router/{ROUTER UUID}
  - Details of a single neutron router including external networking details if a ruoter gateway is set
- /networking-vpp/nodes/{SERVER}/routers/floatingip/{FIP UUID}
  - This etcd entry contains the data required to program 1-to-1 SNAT on a single VPP router.

Layer 3 etcd Key/Value sample

/networking-vpp/nodes/bxb-ds-51.bxb.os/routers/interface/9d8cd87b-09be-4b87-96b8-b29e1e771ae9

- {"segmentation_id": 181, "mtu": 1500, "vrf_id": 1, "gateway_ip": "20.0.0.1", "prefixlen": 24, "net_type": "vlan", "loopback_mac": "fa:16:3e:b9:48:86", "physnet": "physnet1", "is_ipv6": false}

/networking-vpp/nodes/bxb-ds-51.bxb.os/routers/router/9d8cd87b-09be-4b87-96b8-b29e1e771ae9

```
{           "status": "ACTIVE", "external_gateway_info": {"network_id": "c9316ebc-65a9-4b31-b478-18bd9e82a396", "enable_snat": true, "external_fixed_ips": [{"subnet_id": "ef282a5d-01c0-4858-8eaa-fb57386dfb23",
"ip_address": "50.0.0.9"}]}, "external_segment": 133, "description": "", "gw_port_id": "4bc69cb0-4a42-49cf-8d8c-8c5ff3d31628", "admin_state_up": true, "tenant_id": "5d0cd1a146be414f9fc482ba557e035b",
"created_at": "2017-04-20T20:38:50Z", "updated_at": "2017-04-24T16:33:59Z", "name": "r1", "external_physnet": "physnet1", "gateways": [["50.0.0.9", 24]], "revision_number": 12, "vrf_id": 1, "project_id":
"5d0cd1a146be414f9fc482ba557e035b", "id": "9d8cd87b-09be-4b87-96b8-b29e1e771ae9", "external_net_type": "vlan"}
```

/networking-vpp/nodes/bxb-ds-51.bxb.os/routers/floatingip/aaae2779-2713-4e26-a68e-b851af86ff0d

```
{           "internal_segmentation_id": 181, "external_net_type": "vlan", "internal_net_type": "vlan",

            "fixed_ip_address": "20.0.0.7", "floating_ip_address": "50.0.0.4", "external_segmentation_id": 133, "event": "associate", "physnet": "physnet1 » }
```

Networking-vpp: Roadmap

Next version will be networking-vpp 17.07

- Security Groups
  - support for remote-group-ID
  - Support for additional protocol fields
- VXLAN-GPE
  - support for ARP handling in VPP
  - Resync states in case of agent restart
- Improved layer 3 support:
  - support for HA (VRRP based)
  - Resync for layer ports
- Testing, testing, testing