Alternatives for Improving OpenStack Networking to Address NFV Needs

Margaret Chiosi
AT&T Labs Distinguished Network Architect
Open Platform for NFV – OPNFV President (Linux Foundation)

Ian Wells
Principal Engineer, Cisco

Ildikó Váncsa
OpenStack Coordinator, Ericsson
Controller Relationships

**Service Orchestrator**

- **SDN-G**
  - Yang Models
  - Heat Templates
- **Openstack**
  - **SDN-L**
    - BGP-like
- **WAN**
- **TOR**
- **NIC**
- **vRouter**

SDN-G – SDN Global Controller
SDN-L – SDN Local
Alternatives for Improving Openstack Networking to Address NFV Needs

Overall Scenario

• Common Openstack-Controller Interface
  • ML2 setup – type (e.g. L2 overlay only, need to add L3) and mechanism (controller plugins)
  • Multiple Servers running different vendor controller & overlay routing software for L3/L2 VPNs – MPLS VRFs
• Multi-vendor routers in one subnet
• Sharing of common control plane parameters – BGP communities (e.g. Route Targets (RT)) across the multi-vendor controllers
• Controllers and routers communicate directly without going through any gateway for east-west traffic
Alternatives for Improving Openstack Networking to Address NFV Needs

ECMP Load Splitting Case – AnyCast

INGRESS Does load split regardless of Host and
Regardless of external (from WAN GW) or internal from G5)
Need separate RD for any cast end point to segregate traffic

Traffic to Anycast 10.1.1.5 can be load split from either WAN GW or another VM like G5

© 2016 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. The information contained herein is not an offer, commitment, representation or warranty by AT&T and is subject to change
Alternatives for Improving Openstack Networking to Address NFV Needs

Hub and Spoke Case

Tenant 1
10.1.1.0/24
10.3.7.0/24

G1 Hub VRF
RD1 10.1.1.5 IP_OVR1 Label1
RD1 0/0 IP_OVR1 Label1
Label 1 Local IF (10.1.1.5)
RD3 10.3.7.9 IP_OVR1 Label2
RD2 10.1.1.6 IP_OVR2 Label3
RD4 10.3.7.10 IP_OVR2 Label3

G2 Spoke VRF
RD1 0/0 IP_OVR1 Label1
RD3 10.3.7.9 IP_OVR1 Label2

Neutron/Network API Calls
1. Create Network
2. Create VRF Policy Resource
   • Any to Any
   • Any to Any w/ ECMP
   • Hub and Spoke (Hub, Spoke)
3. Create Subnet
4. Create Port
   • w/ Subnet
   • w/ VRF Policy Resource, [H | S]
Alternatives for Improving Openstack Networking to Address NFV Needs

Any to Any Base Case

Tenant 1
10.1.1.0/24
10.3.7.0/24

Tenant 2
10.1.1.0/24

Example Neutron/Network API Calls (multi-vendor OVR and one OVR per host)
1. Create Network
2. Create Network VRF Policy Resource
   1. This sets up that when this tenant is put on a HOST that:
      1. There will be a RD assigned per VRF
      2. There will be a RT used for the common any-to-any communication
3. Create Subnet
4. Create Port (subnet, network vrf policy resource)
   1. This causes controller to:
      1. Create vrf in vRouter’s FIB, or Update vrf if already exists
      2. Install an entry for Guest’s HOST-Route in FIBs of Vrouters serving this tenant Virtual Network
      3. Announce Guest HOST-Route to WAN-GW via MP-BGP

VRF Lets us do:
1. Overlapping Addresses
2. Segregation of Traffic

© 2016 AT&T Intellectual Property. All rights reserved. AT&T, the AT&T logo and all other AT&T marks contained herein are trademarks of AT&T Intellectual Property and/or AT&T affiliated companies. The information contained herein is not an offer, commitment, representation or warranty by AT&T and is subject to change.
Analysis
Does this fit in Neutron?

- Neutron today has an L3VPN extension
- We could add extra functionality to it
- …but maybe it’s time to find a different way?
A clean slate approach

• Neutron’s a good API for providing L2 connectivity
• Every network problem at the edge reduces to some form of L2 domain
• So everything always fits in Neutron
• But L3VPNs are rather L3-centric
• The result is that the interface we need needs to align with Neutron’s approach, and things can get a bit… twisty
Difficulties

• The API works, but maybe isn’t what we’d have come up with in a clean slate environment
• The sample implementation needs to be compatible with the existing OVS control code
• ... And all of this without breaking API backward compatibility
• … And as a user you need a network controller that implements every feature you want
A clean slate approach

• How would we design this interface if we started from scratch?
• How would we integrate that API with what exists today?
And yet...

• Neutron’s very good at its L2 model
• An API with all the necessary concepts
• A backend network controller in ML2 that lets multiple drivers interact to deliver the functionality
• We still want to use it
Learning a lesson

• Clouds have three legs – compute, network, storage
• In storage, we have two very different storage APIs – Cinder, for persistent disks, and Swift, for object storage
• Can we do something similar for networking?
What is Neutron?

• Neutron is the network provider for OpenStack
What is Neutron?

• Neutron is the network provider for OpenStack
What is Neutron?

- Neutron is a network provider for OpenStack
Implementation
Today’s solution

- Neutron uses ‘ports’ for connections to VMs
- It uses ‘networks’ with ‘subnets’ as links between those ports
- The L3VPN API today flags a network as having very different behaviour to usual, but sticks with ports to connect to VMs
How would this work?

- Neutron’s ports are universal – all network backends want to connect to VMs, and ports are the means.
- Neutron’s networks and subnets aren’t universal – they’re specific to what Neutron is particularly good at doing.
- Create a model where – as long as a network provider has ports – it can be used.
- Make sure we can use multiple providers at once.
Gluon

Communication via ports alone

Nova

“Nova for containers”

Neutron

“Neutron for L3VPN”
1: “this is the networking setup I want”

2: “make me VMs”

Gluon

Nova

Find the ports and attach to them

Network API

SDN controller A

Network API

SDN controller B

Network API
The implications

The APIs: simple REST models
Code is a web service – fast request-response
Here, we share API constructs (e.g. the basics of a port) and base code (lots of boilerplate)

The protocol: synchronise desired state from the API objects to the network controller (big problems: fault tolerance, asynchronicity; solve them once and well)

The controller: a choice of implementation
Code is event driven - doing hundreds of things at once
Make common implementations, frameworks
What are we hoping this will achieve?

- More innovation – anyone can design a new API and quickly see how it works out in practice; no need to integrate with a single project and prove you’re not breaking it, and anyone can use what you’ve written
- More deployment choices – can deploy multiple network controllers and get the best of each
- More stability – each bit of code is much simpler
What are the risks we run?

- More proliferation – everyone writing their own different API for the same type of networking
- Less quality – there’s no one implementation everyone’s working on

We need to follow the IETF approach: ‘rough consensus and working code’
**TAKE THE FIRST STEPS**

› What we need
  - NFV-ready cloud platform
  - Standardized APIs that fulfills NFV use cases
  - Flexible solution to integrate different SDN controllers into the platform

› How we get it
  - Find the gaps
  - Identify alternatives to fulfill the requirements
  - Create prototypes, evaluate the candidates
  - Implement the chosen solution

› Do it as a community effort
OPNFV OVERVIEW

Orchestration and Management

Virtual Network Functions

- Compute Virtualization
- Storage Virtualization
- Network Virtualization

OpenStack
- KVM
- Ceph
- OpenDaylight
- OpenContrail
- ONOS
- OVS

Compute
Storage
Network

Data Plane Acceleration
- DPDK
- ODP

Infrastructure
- Pharos Community Labs
- OPNFV Bare Metal Lab

Upstream Project Collaboration:

Integration
- Alignment
- Installers
- Scenarios

Testing
- Functional
- System
- Performance

New Features
- Fault Mgmt
- IPv6
- SFC
- L3VPN
- Reservation

Continuous Integration / Continuous Deployment

Documentation
WHY OPNFV?

- A community that gathers telecom vendors and service providers together
- Gives a clear NFV focus

- Provides a framework
  - Analysis
  - Implementation
  - Integration
  - Testing

- Works together with upstream communities

- Connected to standardization bodies
New OPNFV initiative

Focusing on OpenStack as VIM
- Find the limitations
- Create an enhanced solution

Collaborates with existing networking projects

Leverages OPNFV test frameworks
- Evaluate the prototypes
- Use different SDN controllers as back ends
COME AND JOIN US!

https://wiki.opnfv.org/project_proposals/netready